TP 7708 E

Statpower

AN ADVANCED ON-BOARD BATTERY CHARGER FOR ELECTRIC VEHICLES
- FINAL REPORT -

Prepared for

Transportation Development Centre
Transport Canada

by

AC Statpower Inc.

May 1986

CA1
DTC 801
86
A25F
c.2

TP 7708 E

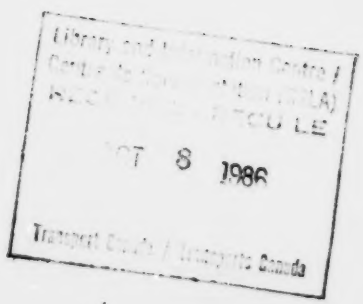
AN ADVANCED ON-BOARD BATTERY CHARGER FOR ELECTRIC VEHICLES

by

Konrad Mauch

AC Statpower Inc.

May 1986



The contents of this report reflect the views of the author and not necessarily the official view or opinions of the Transportation Development Centre of Transport Canada

Un sommaire en francais de ce rapport est inclus avant la Table des matieres.

REPORT DOCUMENTATION FORM

1. Transport Canada Report No.	2. TDC Project No.	3. Recipient's Catalogue No.
TP 7708 E	4442	

4. Title and Subtitle	5. Report Date
AN ADVANCED ON-BOARD BATTERY CHARGER FOR ELECTRIC VEHICLES	MAY 1986
	6. Performing Organization Report No.

7. Author(s)	8. Transport Canada File No.
Konrad Mauch	D1465-351

9. Performing Organization Name and Address	10. DSS File No.
AC Statpower Inc. #308 3700 Gilmore Way Burnaby, B.C. V5G 4M1	17ST.T8200-1-1513
	11. DSS or Transport Canada Contract No.
	1ST81-00064

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
Transportation Development Centre (TDC) 200 Dorchester Blvd. West Montreal, Quebec H2Z 1X4	Final
	14. Sponsoring Agency Code

15. Supplementary Notes	16. TDC Project Officer
	J.H. Morgan

17. Abstract

This report describes the design of a prototype on-board battery charger for electric vehicles. The charger operates from 120 V or 240 V single phase AC power and charges batteries with nominal voltages ranging from 90 V to 108 V. It is designed to charge modern high-energy-density lead-acid traction batteries using a three-step charging profile that follows battery manufacturers' recommendations.

In order to reduce the size and weight of the charger, a switched-mode power conversion circuit is used. The charger is microcomputer controlled. Normal operation of the charger is fully automatic.

An operating manual for the charger is included in an appendix.

18. Key Words	19. Distribution Statement
Battery Chargers, electric vehicles, lead-acid batteries,	Limited number of copies available from the Transportation Development Centre on request.

20. Security Classification (of this report)	21. Security Classification (of this page)	22. No. of Pages	23. Price
Unclassified	Unclassified	141	------

TDC CDT 79 005

Transports Canada / Transport Canada
Développement / Development

FORMULE DOCUMENTAIRE DU RAPPORT

1. N° du rapport Transports Canada TP 7708E	2. N° de l'étude — CDT 4442
3. N° de catalogue du destinataire	
4. Titre et sous-titre An Advanced On-Board Battery Charger for Electric Vehicles	5. Date du rapport Mai 1986
	6. N° du rapport de l'organisme exécutant
7. Auteur(s) Konrad Mauch	8. N° de dossier Transports Canada D1465-351
9. Nom et adresse de l'organisme exécutant AC Statpower Inc. #308 3700 Gilmore Way Burnaby, B.C. V5G 4M1	10. N° de dossier — ASC 17ST.T8200-1-1513
	11. N° de contrat — ASC ou Transports Canada 1ST81-00064
12. Nom et adresse de l'organisme parrain Centre de développment des transports (CDT) Complexe Guy Favreau 200 ouest, boul. Dorchester - Suite 601 Montréal, Québec H2Z 1X4	13. Genre de rapport et période visée
	14. Code de l'organisme parrain
15. Remarques additionnelles	16. Agent de projet — CDT J.H. Morgan

17. Résumé

Ce rapport concerne la conception du prototype d'un chargeur embarcable pour accumulateurs de traction, capable de fonctionner sur un courant 120 ou 240 V alternatif, monophasé et de charger des accumulateurs d'une tension nominale entre 90V et 108V. La charge des accumulateurs plomb-acide à énergie massique élevée, utilisés pour la traction routière électrique s'effectue suivant trois régimes de fonctionnement et respecte les instructions des constructeurs.

La réduction de la masse et du volume du chargeur a été obtenue par l'adoption d'un convertisseur à alimentation à découpage. Il est piloté par un micro-ordinateur qui régule la charge, le fonctionnement restant entièrement automatique.

Les instructions sur la marche du chargeur sont données en annexe au rapport.

18. Mots-clés chargeurs, véhicules électriques, accumulateurs plomb-acide	19. Diffusion Le CDT dispose d'un nombre limité d'exemplaires.

20. Classification de sécurité (de ce rapport) non-classifié	21. Classification de sécurité (de cette page) non-classifié	22. Nombre de pages 141	23. Prix ------

CDT/TDC 79-005 Rev 86

SUMMARY

The battery charger is an important but often neglected component of an electric vehicle. The charger strongly influences the efficiency, battery cycle life, and ease of use of the vehicle. The Transportation Development Centre of Transport Canada has supported work to develop an advanced on-board battery charger with the following features:

- high energy efficiency, better than 85%
- size and weight such that the unit can be installed in a vehicle
- capability to recharge a fully discharged battery overnight
- charging profile that meets battery manufacturers' recommendations for their new high energy density batteries.

A switched-mode power conversion circuit is used in the charger in order to reduce the size and weight. A transformer isolated buck-boost (flyback) circuit topology is used. The flyback circuit can be controlled so that the current drawn form the utility line is approximately sinusoidal and in phase with input voltage. As a result, the input power factor is close to unity and the injection of harmonics into the utility line is fairly low.

The power conversion circuit has the following ratings:

Input Voltage:	100 to 260 VAC, single phase
Input Current:	0 to 20 A rms
Output Voltage:	80 to 140 VDC
Output Current:	0 to 50 A
Maximum Output Power:	5 kW

The power conversion section of the charger is controlled by an Intel 8085 based microcomputer. It receivves signals from sensors which measure the battery voltage, battery temperature, and charging current. The microcomputer uses these feedback signals to control the power conversion circuit so that the proper charging profile is followed.

The charging profile for the battery charger was selected after

reviewing the literature and surveying battery manufacturers for their recommendations. The profile consists of three distinct modes:

1) Constant Current Mode When charging begins, the charger delivers a constant current to the battery. The magnitude of the charging current is usually limited either by the maximum current rating of the charger or by the power available from the utility line.

2) Constant Voltage Mode When the battery voltage rises to the gassing voltage, the charger reduces the charging current to maintain the battery voltage at the gassing voltage.

3) Finishing Charge When the charging current has dropped to approximately 5 amperes, the charger continues charging at that current level for about three hours. This results in a slight overcharge which provides some degree of equalization and prevents problems such as acid stratification and plate sulphation.

This profile charges the battery at the maximum possible rate within the limitations imposed by charger capacity, available utility power, and battery charge acceptance.

The microcomputer performs a number of other functions. It uses the sensor data to detect possible fault conditions during charging. The microcomputer also manages the control panel functions of the charger. The control panel is a small computer terminal that is connected to the microcomputer via an RS-232 serial data link. The computer terminal has advantages over a more conventional dedicated control panel. The terminal can be installed in the passenger compartment of the vehicle and connected to the charger with a simple, readily available cable. The number and type of commands and status indications are limited by the microcomputer software rather than by the number of buttons and lights on a conventional control panel. Thus the user interface can be modified quite easily if required.

When the charger is first installed in the vehicle, the user initializes the charger by entering information such as the time of day, day of week, and type of battery being charged. The user also specifies the time of day when charging should commence and the day of the week and time when an equalization charge should be performed. The microcomputer maintains an internal clock which keeps track of the time of day and the day of the week after it is initialized. It automatically starts charging at the time set by the user. It will also automatically initiate an equalization charge on the day of the week set by the user.

Charging can also be started directly by a command from the user.

The charger control circuits are powered by a small switching power supply which gets its power from the vehicle battery. This allows the microcomputer to run even when the charger isn't connected to the AC line. This is important in order to maintain the correct time. The charger will not operate if it is not connected to a battery or if the battery is so deeply discharged that its voltage has fallen below 70 volts. The power supply only draws 200 to 300 milliamperes of current from the vehicle battery, so its effect on vehicle range is negligible.

The charger is housed in an aluminum cabinet with dimensions of approximately 49 cm x 49 cm x 18 cm. Its weight is approximately 25 kg. The cabinet has sockets for connections to AC power, the battery temperature sensor, and the control panel. The connections to the battery are made through two threaded studs since there is no standard connector for this.

Although the charger can be mounted on board a vehicle, the size and weight of the charger should be reduced further in order to make installation more convenient. Another area requiring further work is the cost of the charger. At present, the manufacturing cost of the charger is too high to make it commercially viable. Further work is required to either reduce the manufacturing cost of the present design or come up with an alternative design approach that is fundamentally less expensive.

SOMMAIRE

Accessoire important, mais souvent négligé dans la traction routière électrique, le chargeur joue un rôle capital: il augmente le rendement, prolong la vie utile des accumulateurs et ajoute à la souplesse des véhicules électriques. Grâce au concours financier du CDT, les travaux de R&D sur un chargeur embarqué de type avancé ont débouché sur les résultats suivants:

- rendement énergétique supérieur a 85 p. 100,
- réduction de la masse et du volume au point d'en permettre l'installation à bord du véhicule,
- capacité de rechargeur en une nuit un accumalateur complètement déchargé,
- modalités de charge répondant aux instructions des constructeurs concernant les accumulateurs modernes à énergie massique élevée.

La réduction de la masse et du volume est obtenue par l'adoption d'un convertisseur à alimentation à découpage, à montage du type à accumulation isolé. Le circuit de puissance est commandé de manière que le courant tiré du réseau soit de forme à peu près sinusoidale et en phase avec la tension. Le facteur rendement de puissance à l'entrée s'approche alors de l'unité et l'injection d'harmoniques dans le réseau reste faible.

Le circuit de puissance possède les caractéristiques suivantes:

Tension a l'entrée	100 à 260 V alternatif, monophasé
Courant efficace à l'entrée	0 à 20 A
Tension à la sortie	80 à 140 V, continu
Courant à la sortie	0 à 50 A
Puissance maximum débitée	5 kW

Le circuit de puissance est piloté par un micro-ordinateur Intel 8085 qui regulé la charge en fonction des signaux qu'il reçoit sur la tension et la température dans les accumulateurs ainsi que sur le courant de charge.

Les modalités de la charge ont été choisies à la suite d'une recherche documentaire et de consultations avec les constructeurs. La charge s'effectue suivant trois régimes:

1) Charge à courant constant. Au début, la charge se fait à courant constant, à une allure déterminée soit par le calibre du chargeur, soit par la puissance fournie par le réseau de distribution électrique.

2) Charge à tension constante. Lorsque la tension de

dégagement gazeux est atteinte, le courant de charge décroît de manière à garder la tension de batterie égale à la tension de dégagement gazeux.

3) Fin de charge. Lorsque le courant de charge tombe à 5 A environ, la charge se poursuit à courant réduit pendant environ trois heures. Il en résult une surcharge qui facilite quelque peu l'égalisation des éléments tout en àidant a prévenir la stratification et la sulfatation.

Ainsi, la charge peut s'effectuer à une allure maximale, compte tenu du calibre du chargeur, de la puissance fournie par le réseau et de l'acceptance de charge.

Le rôle du micro-ordinateur est double: réguler la charge de manière à prévenir les anomalies possibles et piloter la marche du chargeur par l'entremise d'un petit terminal auquel il est relié par un bus série RS-232. Ce terminal possède plusieurs avantages sur le coffret de command classique. Outre qu'il peut trouver place dans la cabine, étant relié au chargeur par un câble simple disponible sur le marché, il multiplie le nombre de commandes et d'indications d'état possibles grâce à une commande programmée beaucoup plus universelle que celle autorisée par les boutons at les voyants d'un coffret de command classique. Ainsi, l'interface conducteur/chargeur devient facile à modifier, selon les besoins.

Lors de l'installation du chargeur à bord du véhicule, on fait son initialisation en chargeant dans sa mémoire diverses informations comme: l'heure, le jour de la semaine et le type d'accumulateur à charger, ainsi que l'heure à laquelle la charge devrait commencer et l'heure et le jour dans la semaine auxquels la charge d'égalisation doit être effectuée. Grâce à une horloge intégrée, le micro-ordinateur obéit automatiquement aux instructions séquentielles. Mais il est également possible de mettre le chargeur en marche par une command direct.

Les circuits de commande sont alimentés à même la batterie du véhicule, par l'intermédiaire de petit interrupteurs. De la sorte, le micro-ordinateur peut fonctionner, même lorsque le chargeur n'est pas sous tension, et rester à l'heure juste. Le chargeur ne fonctionne que s'il est connecté à une batterie et que la tension de celle-ci est d'au moins 70 V. Concernant les circuits de command, ils consomment seulement 200 à 300 mA, ce qui ne compromet pas l'autonomie du véhicule.

Le chargeur est placé dans un coffret en aluminum de 49 x 49 x 18 cm, environ, et sa masse est autour de 25 kg. Des prises permittent de connecter le chargeur au réseau, au capteur de température et au terminal. Deux contacts filetés pemettent de raccorder le chargeur à la batterie, faute de connecteur

standard.

Bien que le chargeur soit embarcable, sa masse et son volume gagneraient à être réduits encore plus, afin d'en faciliter l'emport. Autre facteur à réétudier, le prix du chargeur quie, pour le moment, est encore trop élevé pur trouver sa place sur le marché. Il faudra donc, soit baisser le coût de production du modèle existant, soit étudier un autre modele de réalisation moins chère.

TABLE OF CONTENTS

1. INRODUCTION 1

2. BATTERY CHARGING STRATEGY 3
2.1 Selection of Charging Profile 3
2.2 Determination of Gassing Voltage 5
2.3 Charge Termination 6

3. BATTERY CHARGER POWER CIRCUIT TOPOLOGY 7
3.1 Power Circuit Alternatives 7
3.2 Selection of a Suitable Power Circuit 8
3.3 Power Circuit - Principles of Operation 9

4. OVERVIEW OF BATTERY CHARGER DESIGN 12

5. BATTERY CHARGER POWER CONVERSION CIRCUIT 15
5.1 Input Section 15
5.2 Power Conversion Section 15

6. CHARGER CONTROL CIRCUITS 20
6.1 Inner Loop 20
6.2 Outer Control Loop 23

7. BATTERY CHARGER SOFTWARE 26
7.1 Initialization and Task Scheduling Module ... 26
7.2 Sequencer Module 28
7.3 Control Panel Module 30
7.4 Real-Time Clock Module 30
7.5 Control Loop Module 33

8. AUXILIARY DC/DC CONVERTER 35

9. CONCLUSIONS AND RECOMMENDATIONS 37

REFERENCES 39

APPENDIX A INSTALLATION AND OPERATION MANUAL A-1
A.1 Installation A-1
A.1.1 Physical Installation A-1
A.1.2 Electrical Installation A-1
A.2 Charger Operation A-4
A.2.1 Start-up Procedure A-4
A.2.2 Control Panel A-7
A.2.3 Control Panel Menus A-10
A.2.4 Operating Guidelines A-13

APPENDIX B SCHEMATIC DIAGRAMS B-1

APPENDIX C PROGRAM LISTINGS C-1

LIST OF FIGURES

1. Charging Profile 4

2. Flyback Power Conversion Circuit 10

3. Major Subsystems in the Battery Charger 13

4. Input Section of Power Conversion Circuit 16

5. Battery Charger Power Conversion Circuit 17

6. Battery Charger Control Loops 21

7. Control of Transistor Switching 22

8. Battery Charger Microcomputer System 24

9. Initialization and Task Scheduling Module 27

10. Battery Charger Sequencer State Transition Diagram 29

11. Data Flow for Control Panel Module 31

12. Control Panel Menu Sequences 32

13. Control Loop Module 34

14. DC/DC Converter Operating Envelope 36

15. L6-30P Connector A-4

16. Control Panel Menus A-8

17. Use of Control Panel Commands A-9

18. Charger Input Section B-2

19. Input Voltage Sensing Circuit, Fan Switching Circuit, and Front Panel B-3

20. Transistor Base Drive Power Supply B-4

21. Transistor Base Drive Circuit B-5

22. Charger Power Conversion Circuit B-6

23. Output Crowbar Circuit B-7

24. Inner Control Loop: Digital Circuits B-8

25. Timing Diagram For Digital Section of Inner Control Loop B-9

26. Inner Control Loop: Analog Circuits B-10

27. Waveform Diagram for Analog Section of Inner Control Loop B-11

28. Interface to DAC, Sensor Signals, and Contactor .. B-12

29. Feedback Signal Conditioning Circuits B-13

1. INTRODUCTION

The battery charger is an important but often neglected component of an electric vehicle. The charger strongly influences the efficiency, battery cycle life, and ease of use of the vehicle. In many cases the charger used with an electric vehicle is an off-board unit designed for use with fork lifts or golf carts. For electric automobiles, lighter, more compact chargers that can be installed in the vehicle are desirable.

This report describes work performed by AC Statpower Inc. to develop an advanced on-board battery charger suitable for use on electric automobiles and trucks. The development goals were to develop a prototype charger with the following features:

- high energy efficiency, better than 85%
- size and weight such that the unit can be installed in the vehicle
- capability to recharge a fully discharged battery overnight
- operation from both 120 VAC and 220 VAC outlets
- charging strategy that meets battery manufacturers' recommendations for their new high energy density batteries

The development work began with a review of the literature on the charging of lead-acid batteries and the design of battery chargers for electric vehicles. Battery manufacturers were surveyed for their recommendations on charging their batteries.

From the literature review and the manufacturers' recommendations a number of possible charging strategies were identified. Some of the more promising charging strategies were simulated on a computer to determine recharge times when operated from power outlets with ratings varying from 120 VAC and 12 A to 240 VAC and 25 A. In addition, the charging strategies were analyzed to determine if they could be implemented without the need for complex or expensive circuits or sensors. On the basis of this analysis and the simulations, a charging strategy was chosen for the prototype charger.

In parallel with the development of a charging strategy, a power conversion circuit for the charger was also being developed. A power circuit topology was chosen which met efficiency and size/weight requirements and presented a high power factor to the AC line. A breadboard version of the circuit was constructed to verify its performance. On the basis of results of tests on the breadboard, a revised circuit was designed and constructed for

the prototype charger.

The next phase in the project was to develop the control circuits for the charger. A decision was made early in the program to base the charger controller on a microcomputer and to perform as much of the control task as possible with software. The control of the charging profile is performed by software in the microcomputer as is the overall sequencing and timing of charger operation. Software also carries out the control panel functions for the charger. This software based approach allows the flexibility to adapt to new batteries and to differing user requirements.

Three prototype battery chargers were built as part of the development program. Two were delivered to Transport Canada for testing and evaluation. At Transport Canada's suggestion, one unit will be kept at AC Statpower's facilities for testing and further development work.

As an adjunct to the battery charger development work, a DC/DC converter that supplies 12 volt auxiliary power from the vehicle propulsion battery was also designed. This converter ensures that the auxiliary electrical system (i.e. vehicle lights, windshield wipers, fans, etc.) will have power whenever there is still power in the propulsion battery and eliminates the need for a large auxiliary battery and its associated charger. Three prototypes of this DC/DC converter were constructed and delivered to Transport Canada.

This report first reviews the battery charging strategy used in the charger. The power conversion circuit topology for the battery charger is discussed next. The design of the various component systems of the prototype charger is then discussed and conclusions and recommendations about this development project are presented. The appendices contain an operating guide for the charger and more detailed design information on the charger.

2. BATTERY CHARGING STRATEGY

The charging strategy for a battery charger determines the profile of output voltage and current versus time for the charger. This charge profile must be chosen to meet the requirements of the battery. For the purposes of this project, the batteries to be charged were restricted to lead-acid traction batteries. The charging profile must also be chosen so that the current drawn from the AC line is less than the rating of the outlet yet great enough that the battery can be recharged overnight.

2.1 Selection of Charging Profile

A variety of charge profiles have been used or proposed for use with lead-acid batteries. Some of these, such as a simple constant current or constant voltage profile, are not suitable for rapid recharging. Others, such as certain pulse charging schemes proposed for rapid recharging [1,2] have been shown to have no particular advantage in terms of charging time and to be less efficient than more conventional charging strategies [3].

The most commonly used profile for traction batteries is probably the taper charge. In this profile the output current starts off at a maximum value determined by the difference between the battery voltage and the battery's gassing voltage. The maximum current is also limited by the capacity of the charger. As charging continues, the current remains proportional to the difference between the battery voltage and the gassing voltage. As a result the charging current falls, or tapers, as the battery voltage rises during charging. When the battery voltage has reached the gassing voltage, a voltage sensitive circuit starts a timer which allows charging to continue at a low current for a preset interval before the charger shuts off.

A three-step profile which improves on the taper charge has been recommended by Exide and Globe Union for their high energy density traction batteries. The profile, shown in Figure 1, starts with a constant current charge at a rate determined by the charger capacity. When the gassing voltage is reached, the profile switches to a constant voltage mode where the charger voltage is held at, or slightly below, the gassing voltage. As a result the charge current begins to fall. When the current has dropped to a low value (5 or 6 amperes), the profile enters another constant current charge mode and continues charging at this current until a fixed period of time has elapsed or sufficient ampere-hours have been returned to completely recharge the battery. This approach has the advantage that the battery is charged at the maximum possible rate within the limitations imposed by charger capacity and battery charge acceptance. The

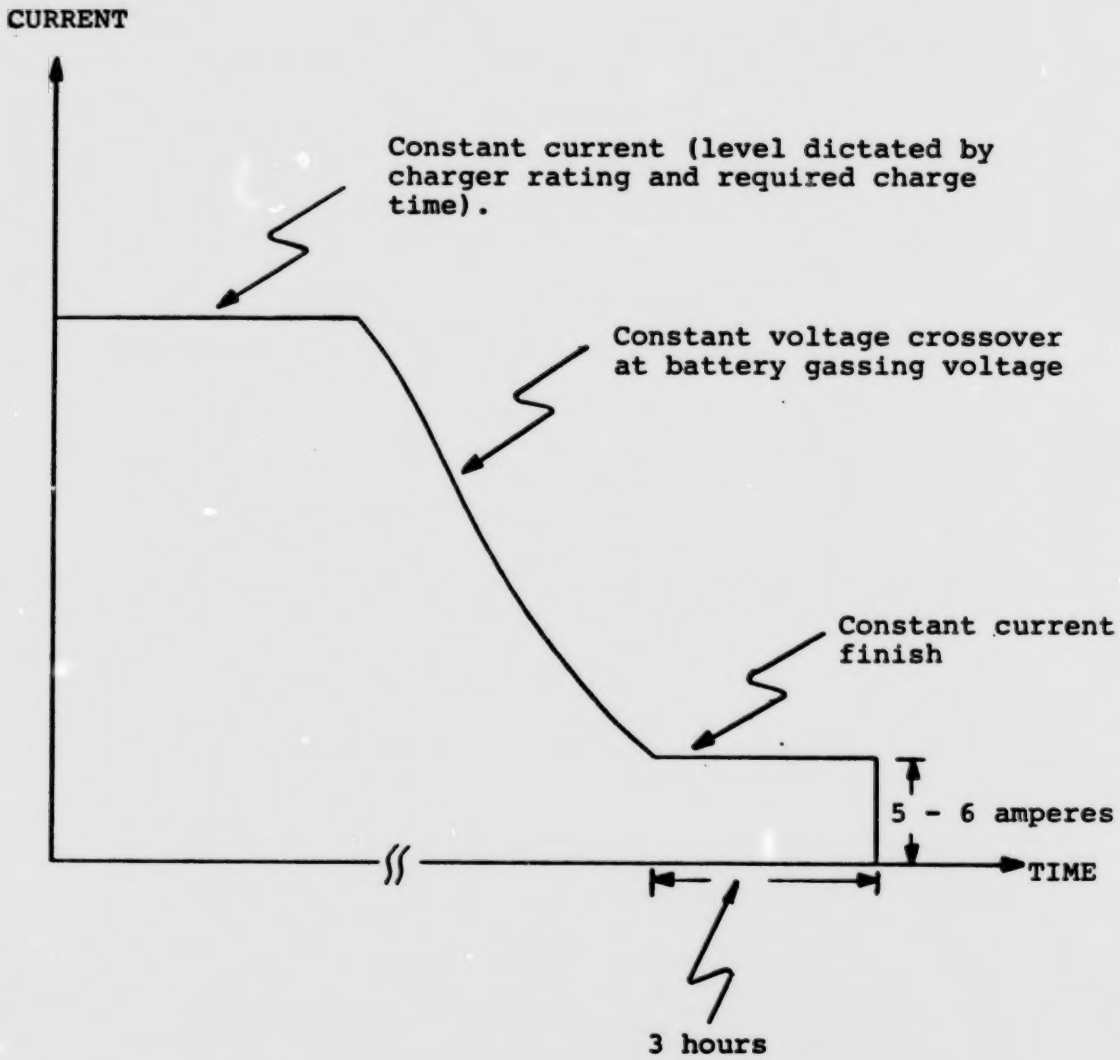

CURRENT
Constant current (level dictated by charger rating and required charge time).
Constant voltage crossover at battery gassing voltage
Constant current finish
5 - 6 amperes
TIME
3 hours

Figure 1. Charging Profile

low current finishing charge at the end provides some degree of equalization and, by allowing some gassing, prevents problems such as acid stratification [4] and plate sulfation which may occur if the battery isn't somewhat overcharged. This profile was adopted for the prototype charger.

2.2 Determination of Gassing Voltage

The charging profile requires that the charger output voltage track the battery gassing voltage during the constant voltage portion of the profile in order to minimize gassing. This presents the following problems:

a) the gassing voltage differs depending on the formulation of the battery plates

b) the gassing voltage varies with electrolyte temperature

c) the gassing voltage changes as the battery ages

One way of avoiding these problem is to monitor the evolution of gas from the battery and then control the charging voltage to maintain the gas flow rate at a low value [5,6]. Since this technique controls battery gassing directly, there is no need to determine the gassing voltage. However, this technique is difficult to implement outside the laboratory. We chose instead to try to compensate for the problems in determining the gassing voltage.

The variation in gassing voltage among different types of lead-acid batteries is dealt with by adjusting the charger for the different battery types. This is relatively easy to do with a microcomputer controlled charger since the gassing voltages for different batteries can be stored in the microcomputer's memory.

The change of gassing voltage with electrolyte temperature is dealt with by measuring the battery temperature and then correcting for the temperature effect on the gassing voltage. Since the temperature coefficient varies somewhat with battery type, the microcomputer actually stores a different gassing voltage equation for each different battery type.

At present, only two different equations are stored since only two battery manufacturers responded to our request for information. One equation is used for the Globe Union EV-1000 battery. The other equation is used for the Exide EV-106 and Exide XPV-23 batteries. It is also used for batteries from other manufacturers since it comes close to the gassing voltage and temperature coefficient commonly cited for lead-acid batteries. The equations for a single three cell battery are:

(1) Globe Union $V = 8.58 - .022(T)$

(2) Exide $V = 7.6 - .018(T)$

where:

V is the charging voltage for the constant voltage portion of the profile

T is the battery electrolyte temperature in Celsius

The change in gassing voltage with battery age proved difficult to deal with. There are references to this change in the literature which suggest that the gassing voltage drops by 20 to 50 millivolts per cell over the life of the battery [7,8]. However the data is scanty and no curves are available to show how the voltage drops as a function of age. Battery manufacturers don't supply any data on how the gassing voltage changes with age either. We decided not to include any correction for aging in the gassing voltage equation in the first version of the control software for the charger. If some hard data on this effect becomes available, the software can be modified to incorporate the aging effect relatively easily.

2.3 Charge Termination

Another factor that must be considered with this charging profile is the charge termination technique. It is desirable to terminate charging after the battery has been recharged to ~110% to 115% of its ampere-hour capacity. This ensures that the battery is never undercharged and produces just enough of an overcharge to ensure acid mixing without causing severe gassing and water loss.

The approach used in this charger is to terminate charging 3 hours after the charger has entered the final constant current portion of the profile. This will produce good results with batteries that are moderately or completely discharged. However, if the charger took less than one hour to complete the first two portions of the profile, the charger terminates charging within a few minutes of entering the final portion of the profile. This ensures that fully charged or slightly discharged batteries are not overcharged.

3. BATTERY CHARGER POWER CIRCUIT TOPOLOGY

The charger power circuit must convert the incoming alternating current from the power line to direct current and then control the outgoing charging current so that it follows the charging profile. In addition, the supply voltage must usually be matched to the battery voltage.

3.1 Power Circuit Alternatives

The most common power circuits for battery chargers use a 60 Hz transformer to match the supply voltage to the battery voltage. Often the transformer is a ferroresonant type that makes use of the non-linear magnetic characteristic of the transformer core to achieve regulation of the secondary voltage. The regulation characteristic of the ferroresonant transformer inherently results in a taper charge profile. Thus the only additional circuits required are a rectifier in the secondary circuit to produce direct current and control circuits to terminate charging. Alternatively, a standard linear transformer may be used, followed by an SCR (silicon controlled rectifier) bridge circuit that provides regulation and rectification.

Although a transformer is usually required to match the supply voltage to the battery voltage and to provide electrical isolation from the AC line, a standard transformer operating at the 60 Hz AC line frquency is very heavy and bulky. By raising the frequency at which the transformer is operated, the transformer can be made much smaller and lighter. This is the technique used in switched mode battery chargers. The incoming AC current is first rectified to produce direct current. The direct current is then converted to high frequency AC current (typically 20 kHz to 100 kHz) by semiconductor switches and passed through a relatively small transformer. The high frequency current from the secondary of the transformer is then rectified to produce direct current which charges the battery. Regulation of the charging current is usually carried out by the same circuit that converts the input current to high frequency AC current.

Switched mode battery chargers are relatively complex and incorporate a considerable amount of relatively new technology. As a result they are currently more expensive and less reliable than charger designs based on 60 Hz transformers which have been used in industry for 30 years or more. However they offer the advantages of greatly reduced size and weight, higher efficiency and power factor, and greater flexibility in the choice of charge profile. As semiconductor costs decline and more experience is gained with switched mode designs, the disadvantages of switched mode battery chargers should disappear. Since the advantages offered by the switched mode battery charger are very important

for an on-board charger, we decided to use a switched mode power circuit in the protoype charger.

3.2 Selection of a Suitable Power Circuit

There are a variety of switched mode power conversion circuits that are suitable for use in a battery charger. Van Eck et. al. [9] discuss alternate switched mode power circuits and present details of a forward converter circuit that they used in a 3 kW charger that could be used in on-board applications. Gosden [10] describes a series resonant power conversion section used in a 1.5 kW charger designed for on-board use. Rippel [11] discusses the advantages of boost converter power circuits but doesn't appear to have actually built a charger using this approach. However, Latos and Bosack [12] describe a 3 kW on-board charger built by Gould laboratories which uses a boost converter circuit.

Boost or buck-boost converter circuits, which can produce an output voltage which is higher than the input voltage, have particular advantages for battery chargers operating from single phase AC power. When single phase AC power is rectified, the resulting DC voltage varies from zero to the peak value of the AC voltage during every cycle of the AC voltage. A boost or buck-boost converter can boost the DC voltage so that it is greater than the battery voltage for the entire AC cycle, allowing power to be transferred from the AC line to the battery continuously. If a buck converter (which can only produce an output voltage less than the input voltage) is used, the DC voltage will be less than the battery voltage for some portion of every AC cycle. During that period no power will be transferred from the AC line to the battery.

Maximum use is made of the current capacity of the AC power line when power is transferred continuously from the AC line to the battery. This is important since the capacity of the AC power line is often the limiting factor in how quickly a battery can be recharged. A boost or buck-boost power circuit can be controlled so that the current drawn from the AC line is simusoidal and in phase with the AC line voltage. As a result, the battery charger looks like a resistive load to the AC line. This maximizes the power factor and minimizes the injection of harmonic currents into the line; two features which are desirable from the utility's standpoint. Because of these advantages, we decided to use a boost or buck-boost power conversion circuit.

The basic ratings for the power conversion circuit were then defined. They are listed below.

Input Voltage:	100 to 260 VAC, single phase
Input Current:	0 to 20 A rms

Output Voltage: 80 to 140 VDC

Output Current: 0 to 50 A

Maximum Output Power: 5 kW

These ratings allow the charger to be used from both 120 VAC and 240 VAC sources and allow it to charge batteries with nominal voltages of 90 to 108 volts (15 to 18 six volt batteries in series). The 50 ampere maximum output current reflects the recommended maximum charging current for many new high energy density lead-acid batteries.

With the basic power circuit ratings determined, various boost and buck-boost circuit configurations were examined to see if they would be able to meet the requirements. The wide range of input and output voltages made a straight boost configuration impractical. In order to accomodate this range, the output of the boost converter would have to range from 370 volts to 650 volts before being fed through the transformer. At the time this design was initiated, it was not possible to obtain reasonably priced power transistors which could switch 650 volts at the power levels required.

We decided to use a buck-boost (flyback) circuit since it could be designed using transistors with lower voltage ratings. In order to meet the output power requirements, two parallel power conversion circuits are used.

3.3 Power Circuit - Principles of Operation

Figure 2 shows the basic elements of one of these circuits. The circuit is fed by a varying DC voltage from a rectifier connected to the AC line. When the two switches (which are actually power transistors) are closed, current flows through the primary winding of the transformer. The rate of rise of current is limited by the magnetizing inductance of the transformer. The secondary of the transformer is connected so that the rectifier diode is reverse biased while the switches are on. Therefore no current flows in the secondary and all the energy transferred to the transformer is stored in the transformer's magnetic field. When the two switches open, the magnetic field in the transformer begins to collapse. This causes the voltages on the primary and secondary windings to reverse polarity. The rectifier diode on the secondary of the transformer is now forward biased and so current flows in the secondary circuit, transferring the energy stored in the transformer's magnetic field to the battery.

The switches are operated at a frequency of approximately 20 kHz. During each cycle, energy is first transfered from the AC line to the transformer while the switches are on, and then transferred from the transformer to the battery while the switches are off.

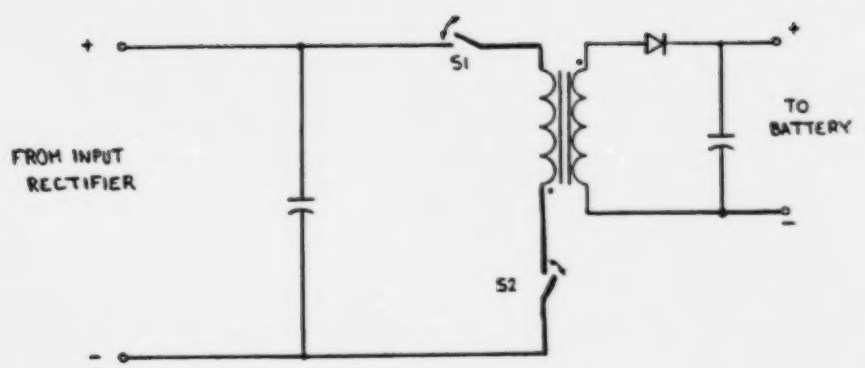


Figure 2. Flyback Power Conversion Circuit

The power transferred to the battery is controlled by varying the ratio of the time the switches are on to the time the switches are off (the duty cycle). The circuits to control the duty cycle are discussed in Section 6 of this report. The power circuit is discussed in more detail in Section 5.

4. OVERVIEW OF BATTERY CHARGER DESIGN

Figure 3 shows a block diagram of the major subsystems in the prototype battery charger. When the charger is charging the battery, the input contactor is closed, connecting the AC supply to the charger. Current from the AC line is rectified and then passed through a filter circuit to the two parallel power converters. The filter prevents high frequency electrical noise from the power converters from entering the AC line. The power converters transfer the power through transformers to the battery.

The converters are controlled by a microcomputer. It receives signals from sensors which measure the battery voltage, battery temperature, and the charging current. The microcomputer compares the actual charging current to the value of charging current required by the charging profile and then sends a control signal to the power converter control circuits to force the charging current to the required value. During the constant voltage portion of the charging profile, the microcomputer compares the battery voltage to the calculated value of the gassing voltage and then adjusts the charging current to keep the battery voltage at or just below the gassing voltage. The gassing voltage is calculated using the information from the battery temperature sensor.

Data from the sensors is also used by the microcomputer to detect possible fault conditions. The battery voltage is compared to upper and lower limits corresponding to 3 volts per cell and 1.6 volts per cell respectively. If the voltage is outside these limits, charging is terminated on the assumption that one or more of battery's cells are damaged. The battery temperature is compared to an upper limit of 50 C and a lower limit of 0 C. Charging is terminated if the battery temperature is outside the limits. A battery temperature in excess of 50 C during charging, assuming an ambient temperature below 40 C, indicates that the battery is defective or is being severely overcharged due to a malfunction in the charger's control system. A battery temperature below 0 C makes charging extremely inefficient. Battery temperature should be maintained above 0 C with heaters and insulating blankets in environments where the ambient temperature is below 0 C.

The control panel for the charger is a small computer terminal that is connected to the microcomputer via an RS-232 serial data link. The computer terminal has advantages over a more conventional dedicated control panel. The terminal can be installed in the passenger compartment of the vehicle and connected to the charger with a simple, readily available cable. The number and type of commands and status indications are limited by the microcomputer software rather than by the number

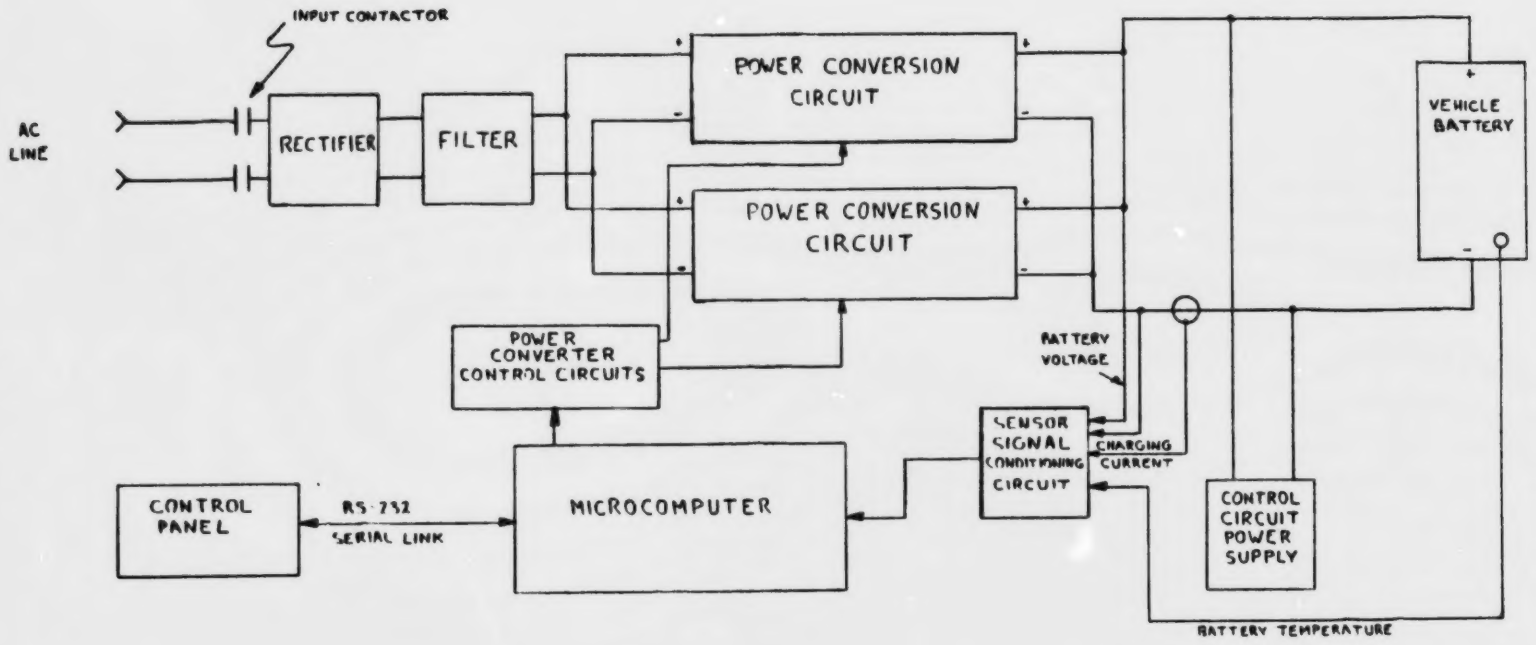


Figure 3. Major Subsystems in the Battery Charger

of buttons and lights on a conventional control panel. Thus the user interface can be modified quite easily if required.

When the charger is first installed in the vehicle, the user initializes the charger by entering information such as the time of day, day of week, and type of battery being charged. The user also specifies the time of day when charging should commence and the day of the week and time when an equalization charge should be performed. The microcomputer maintains an internal clock which keeps track of the time of day and the day of the week after it is initialized. It automatically starts charging at the time set by the user. It will also automatically initiate an equalization charge on the day of the week set by the user. Charging can also be started directly by a command from the user.

The charger control circuits are powered by a small switching power supply which gets its power from the vehicle battery. This allows the microcomputer to run even when the charger isn't connected to the AC line. This is important in order to maintain the correct time. The charger will not operate if it is not connected to a battery or if the battery is so deeply discharged that its voltage has fallen below 70 volts. The power supply only draws 200 to 300 milliamperes of current from the vehicle battery, so its effect on vehicle range is negligible.

The charger is housed in an aluminum cabinet with dimensions of approximately 49 cm x 49 cm x 18 cm. Its weight is approximately 25 kg. The cabinet has sockets for connections to AC power, the battery temperature sensor, and the control panel. The connections to the battery are made through two threaded studs since there is no standard connector for this.

5. BATTERY CHARGER POWER CONVERSION CIRCUIT

5.1 Input Section

A schematic diagram of the input section of the power conversion circuit of the battery charger is shown in Figure 4. The small transformer across the AC line at the input is used to produce a low voltage signal proportional to the input voltage. This signal is used by the control circuit to determine whether the charger is connected to a 120 volt source or a 240 volt source. The signal also provides a reference which is used to keep the input current sinusoidal and in phase with the input voltage.

The AC input is connected to a contactor which isolates the remainder of the charger from the AC line when charging is not in progress. The contactor is controlled by the microcomputer. Cooling fans for the power conversion section are connected after the contactor. Therefore, they only operate when charging is in progress. The fans are 120 VAC units which are connected across the AC line in parallel when the input voltage is 120 VAC and are connected in series when the input voltage is 240 VAC. This is done automatically by a small relay which is controlled by the input voltage sensing circuit.

AC power destined for the power conversion circuits then passes through an EMI (electomagnetic interference filter) which acts to limit the injection of high frequency electrical noise generated by the power conversion circuits into the AC line. The AC power is then rectified by a bridge rectifier to produce a pulsating DC voltage. Some of the power is drawn off through a diode and filtered by a capacitor to produce a relatively constant DC voltage which is used as a source for the power supplies for the transistor base drive circuits in the power conversion circuits. The bulk of the power passes through an inductor-capacitor filter. This filter provides further attenuation of the high frequency electrical noise generated by the power conversion circuits but does not filter out the 120 Hz pulsations in the DC voltage. This pulsating DC voltage is fed to the two power conversion circuits in the charger.

5.2 Power Conversion Circuit

The schematic diagram for the power conversion circuit is shown in Figure 5. The circuit is a flyback type buck-boost converter with some special features to enable it to transfer up to 2.5 kW of power. General information on the design of flyback converters can be found in the text by Chryssis [13]. The discussion here will focus on the features particular to this

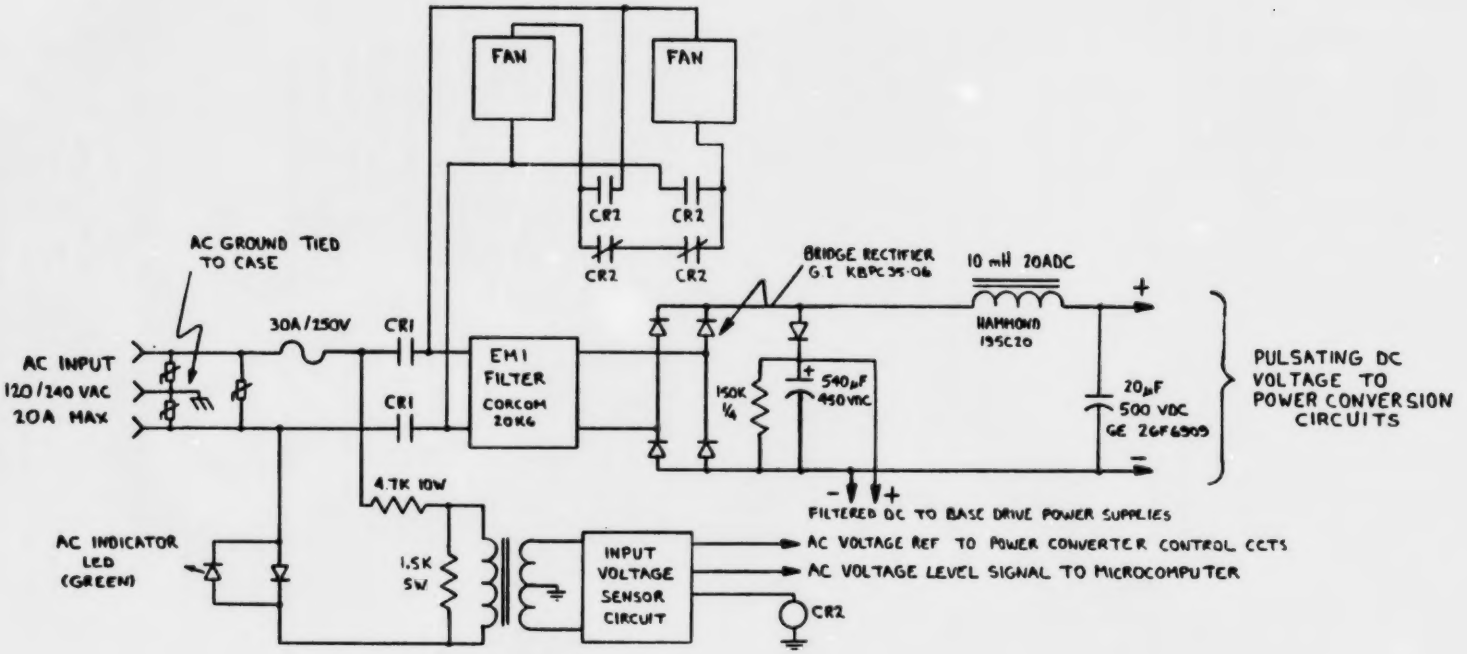


Figure 4. Input Section of Power Conversion Circuit

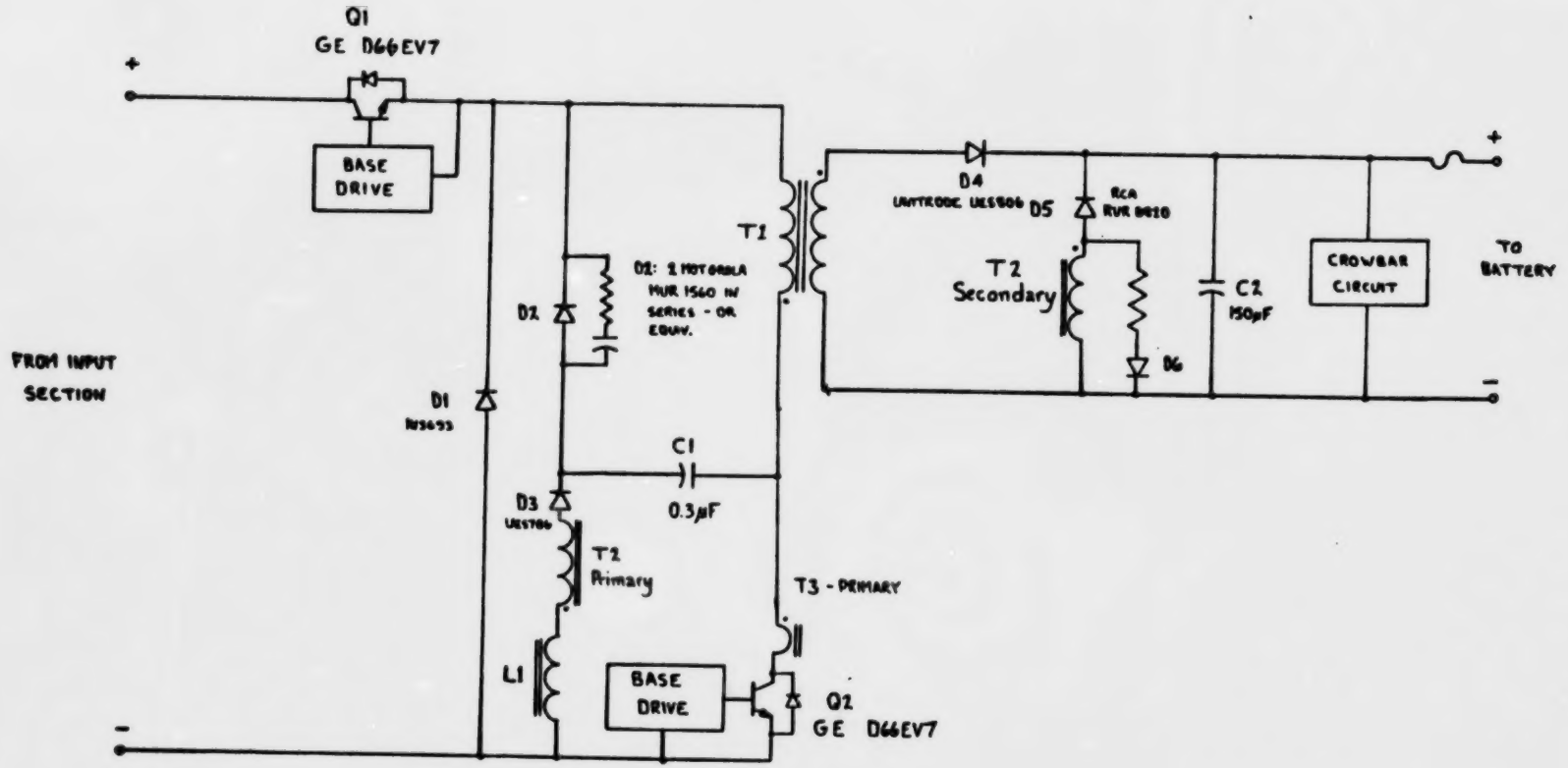


Figure 5. Battery Charger Power Conversion Circuit

converter.

Two transistors are used in this circuit to switch the transformer primary current. Transistor Q1 blocks the input voltage when current is switched off, while transistor Q2 blocks the reflected secondary voltage which appears across the primary of transformer T1. This approach was taken because no single transistor could be found that had a sufficiently high blocking voltage combined with the required current capacity and switching speed.

The input voltage will be at a maximum when the charger operates on a 240 volt circuit. Assuming that the line voltage can be as much as 10% higher than the nominal voltage, the peak input voltage will be

$$240 \text{ VAC} \times 1.1 \times 1.414 = 373 \text{ volts}$$

Q1 must be rated to block at least this voltage.

Transformer T1 has a 3:1 turns ratio. Assuming an ideal transformer, the maximum voltage that Q2 must block should be

$$3 \times 140 \text{ VDC} = 420 \text{ volts}$$

However the transformer is not ideal. In particular, the leakage inductance of the transformer strongly affects the performance of the circuit. It acts to delay the transfer of current from the primary to the secondary winding when Q1 and Q2 switch off. This results in a large voltage appearing across the primary winding (and thus across Q2) as the magnetizing inductance attempts to force current to flow.

This problem is dealt with in two ways. First, T1 is designed for low leakage inductance through the use of interleaved primary and secondary windings and by choosing a core geometry that minimizes the height of the windings [14]. Second, a non-dissipative snubber circuit, consisting of C1, L1, T2, D3, and D2, is used to limit the voltage transient when Q1 and Q2 switch off. The snubber circuit also improves the current vs. voltage locus for Q1 and Q2 when they switch off and thus reduces transistor stress and losses. The energy absorbed by the snubber circuit is transferred to the load through T2 or back to the input DC bus via D2 and the emitter-collector diode in Q1. Whitcomb [15] discusses snubber circuits of this type in more detail.

Transistors Q1 and Q2 are driven by base drive circuits designed to minimize the switching times of the transistors. The base drive circuits receive control signals from the power converter control circuits via optical isolators. Power for the base drive circuits comes from switching power supplies which operate from

the input bus. Therefore, the base drive circuits are only active when the input contactor is closed.

The inner control loop for the power converter circuit controls the peak current flowing through transistors Q1 and Q2. This current is sensed by the current transformer T3. Since the transistor current is controlled directly, the circuit is inherently protected against short circuits. However, an open circuit in the load will cause the converter to produce a very high output voltage as it tries to force current into the load. The microcomputer does not react quickly enough to protect the converter. Therefore a crowbar circuit is used to short-circuit the output if the output voltage exceeds 160 volts. The microcomputer will then shut the charger down.

The power conversion circuit ranges in efficiency from close to 95% at the beginning of the charge when it is operating at maximum output power to about 75% at the end of charge when it is delivering minimum power. The reason for the drop in efficiency is that fixed losses, such as the transistor base drive power consumption become a larger fraction of the total input power at lower power levels. Since most of the energy is delivered to the battery during the early portion of the charge profile, the energy efficiency of the charger should meet the 85% goal.

6. CHARGER CONTROL CIRCUITS

The control loops for the battery charger are shown in Figure 6. They consist of an inner loop which controls the peak current flowing through the transistors in the power converters, and an outer loop which controls the charger output voltage or current.

6.1 Inner Loop

In the inner loop, the input voltage from the AC line is transformed to a low voltage and rectified to form a current waveform command signal. This signal is multiplied by a current amplitude command signal from the microcomputer to produce the transistor peak current reference. The amplitude command signal determines the average value of the current while the current waveform command signal controls the current waveform so that the charger input current will be approximately sinusoidal and in phase with the input voltage.

Figure 7 shows how transistor switching in a power converter is controlled. The switching cycle is started every 50 microseconds by a pulse from a 20 kHz oscillator which sets the flip-flop controlling transistor switching. Current then begins to flow in the transistors. The current has a ramp and pedestal waveform. A current signal is fed back to the control circuit via a current transformer (T3 in Figure 5). When the ramp intersects the transistor peak current reference, a comparator changes state, resetting the flip-flop and thereby causing the transistors to switch off. This control loop provides cycle-by-cycle control of the peak current in the transistors. When an outer loop is closed around this inner loop, the output voltage or current of the converter can also be controlled.

Power converters which use this type of current control scheme are subject to a form of instability when the duty cycle is above 50% [16]. To stabilize the converter, an additional ramp waveform is usually added to the transistor peak current reference. This stabilization technique is used in the control circuits for the battery charger power conversion circuits.

The control circuit doesn't actually switch the two transistors in a power circuit simultaneously. When the devices switch on, the lower device (Q2) is switched on first. When the devices are switched off, the upper device (Q1) is switched off first. This sequence assures proper operation of the snubber network.

The two power conversion circuits in the battery charger are controlled so that the transistors in one circuit are switched on 25 microseconds after the transistors in the other circuit. As a result, the combination of two power conversion circuits appears

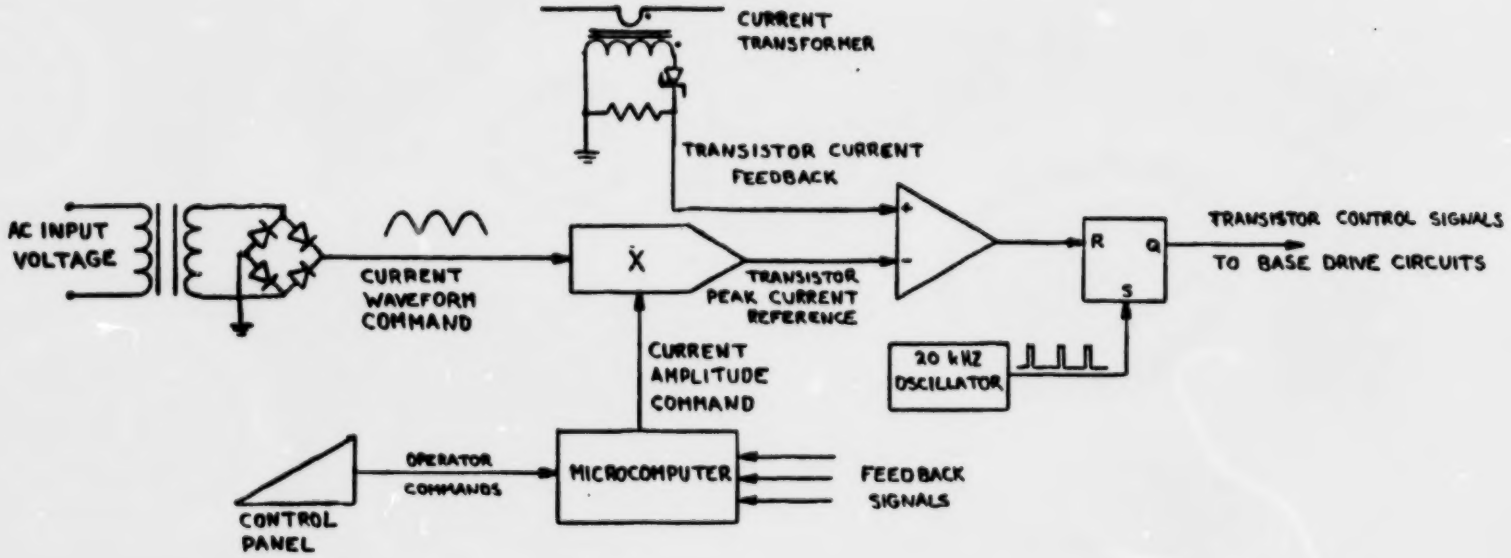


Figure 6. Battery Charger Control Loops

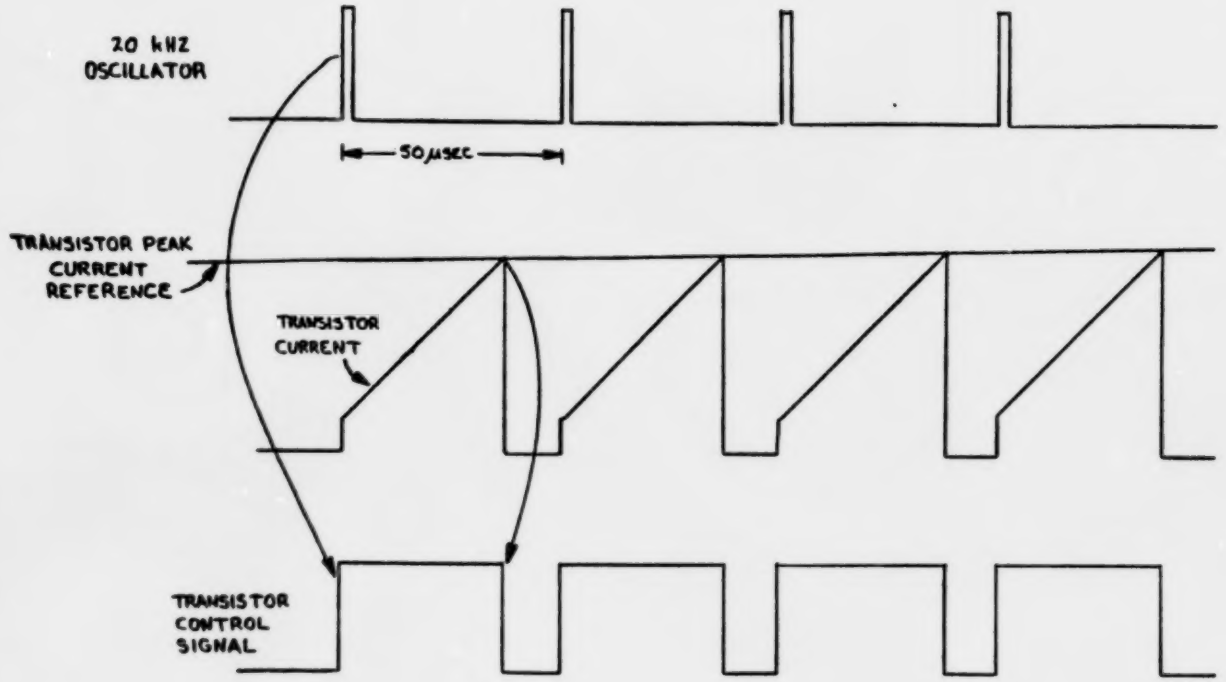


Figure 7. Control of Transistor Switching

to be switching at 40 kHz rather than 20 kHz. This allows smaller filter components to be used at the input and output of the charger.

6.2 Outer Control Loop

The outer control loop is responsible for controlling the charger's output so that the battery charging profile is followed. This is done by the charger's microcomputer which adjusts the current amplitude control signal to force the output voltage or current to the required value.

Figure 8 shows the basic elements of the microcomputer system. The microcomputer receives feedback signals proportional to the battery voltage, charging current, and battery temperature. The battery voltage is measured by a voltage divider across the output terminals of the charger. The charging current is measured by a current shunt in the negative output line of the charger. Battery temperature is measured by a solid-state temperature sensor which is mounted on the battery. A cable and plug connect the sensor to the charger.

The feedback signals are converted from voltage levels to pulse trains by voltage to frequency converters. The frequency of the pulse train is proportional to the voltage level. The pulse train signals are fed to a multiplexer which allows the microcomputer to send any one of the three signals to Counter #2 in the 8253 Counter/Timer IC. The microcomputer reads the count that has accumulated in this counter after a fixed time period to get a binary number proportional to the original voltage level. This conversion technique is rather slow, requiring approximately 100 milliseconds, but is very insensitive to electrical noise and is relatively inexpensive.

The microcomputer uses the information from the feedback signals to determine what corrections must be made to the current amplitude command in order to follow the charging profile. The current amplitude command is an 8 bit binary number which is sent to a multiplying digital-to-analog converter (DAC). This multiplies the current waveform command (an analog signal) with the digital amplitude command to produce the transistor peak current reference (an analog signal).

The microcomputer is based on commercially available cards for the STD backplane bus. The Intersil ISB-3111 microcomputer board [17] contains the microprocessor (Intel 8085), RAM and EPROM memory, and the counter/timer circuit. The Mostek MDX EPROM/UART board [18] contains the serial interface to the portable terminal used as a control panel. Two other custom built STD bus cards contain the circuits for the inner conrol loops. The four STD bus boards are plugged into a four position STD bus motherboard

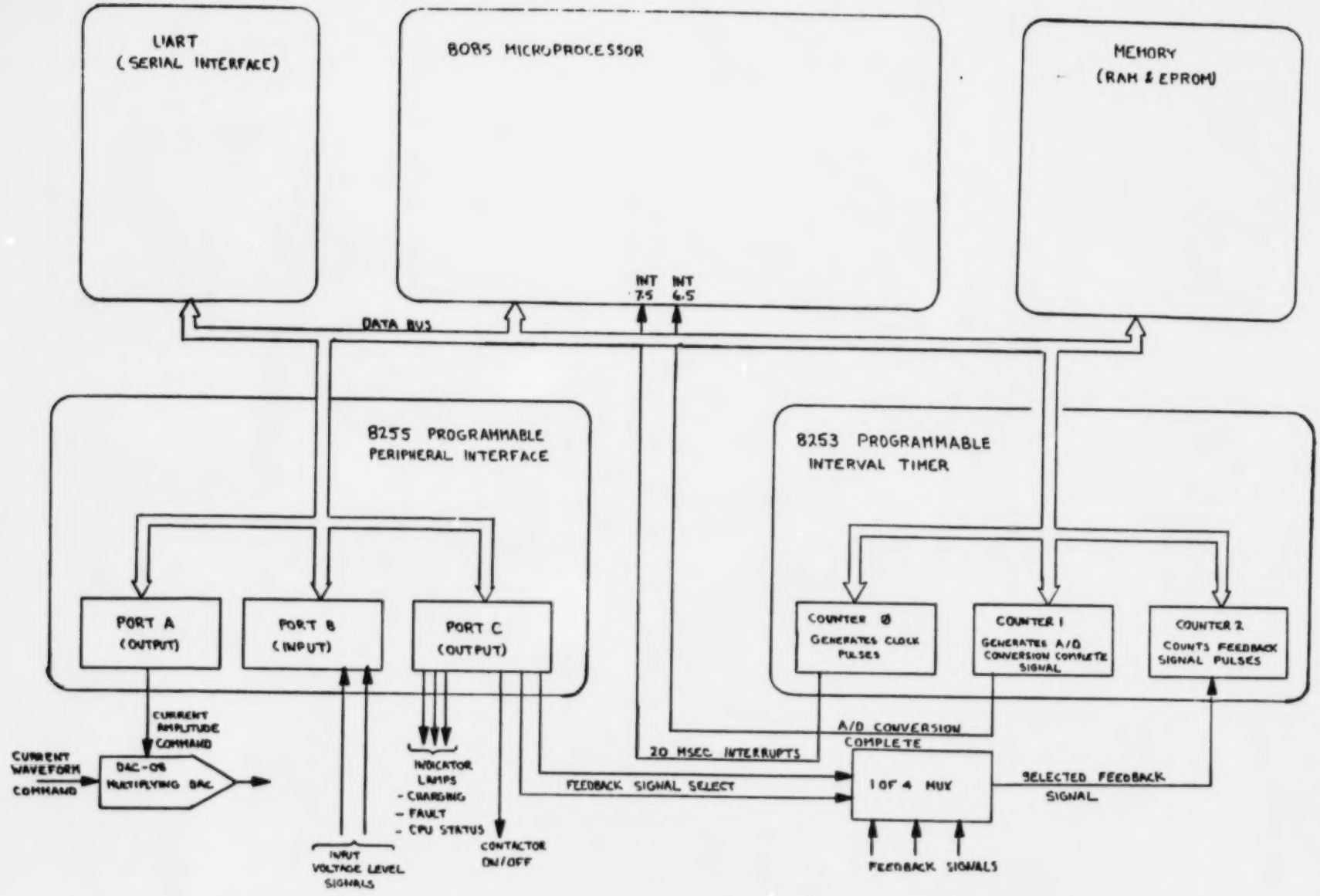


Figure 8. Battery Charger Microcomputer System

and card cage.

The use of commercially available microcomputer boards was cost effective for prototype development. However, a production version of the charger would use a single custom control board. A modern 16 bit single chip microcomputer such as the Intel 8096 would replace the Intel 8085 based microcomputer. The 8096 has the following features [19]:

- On-chip read-only memory (ROM) for program storage
- On-chip read/write memory (RAM) for data storage
- On-chip serial communications interface
- On-chip counter/timers
- On-chip analog-to-digital converters

Therefore a single 8096 chip plus a few additional components can replace the two STD bus microcomputer boards used in the prototype. The number of components required to perform the inner control loop functions can also be reduced by using recently introduced integrated circuits.

7.0 BATTERY CHARGER SOFTWARE

The program to control the battery charger is written primarily in the C programming language [20] with the remainder written in Intel 8085 assembly language. The compiled program occupies about 20 Kbytes of EPROM. The program can be divided into five major modules:

1) Initialization and task scheduling

This initializes data structures, variables, and hardware control registers when the microcomputer is turned on or reset. It also schedules the execution of the following two modules.

2) Sequencer

This controls the overall operation of the charger. It sequences the charger through a number of internal states as the charger performs its functions.

3) Control Panel

This handles communications with the human operator. It allows the operator to set charger parameters, display measured values such as battery voltage and temperature, and control charger operations. A menu based command interface is used.

4) Real-Time Clock

This keeps track of the time of day and day of week. It also times the charging cycle.

5) Control Loop

This acquires the sensor data and controls the current amplitude command to follow the charging profile.

Each of the modules will be described in the following sections.

7.1 Initialization and Task Scheduling Module

Figure 9 shows the basic sequence of operations in this module. The module is entered when the microcomputer is powered up or reset by the reset pushbutton on the charger cabinet. Internal variables and data structures are set to defined initial values. Hardware control registers are initialized so that the charger is in an inactive state. The contactor is opened and the current amplitude command is set to zero. All the indicator lamps controlled by the microcomputer are turned off.

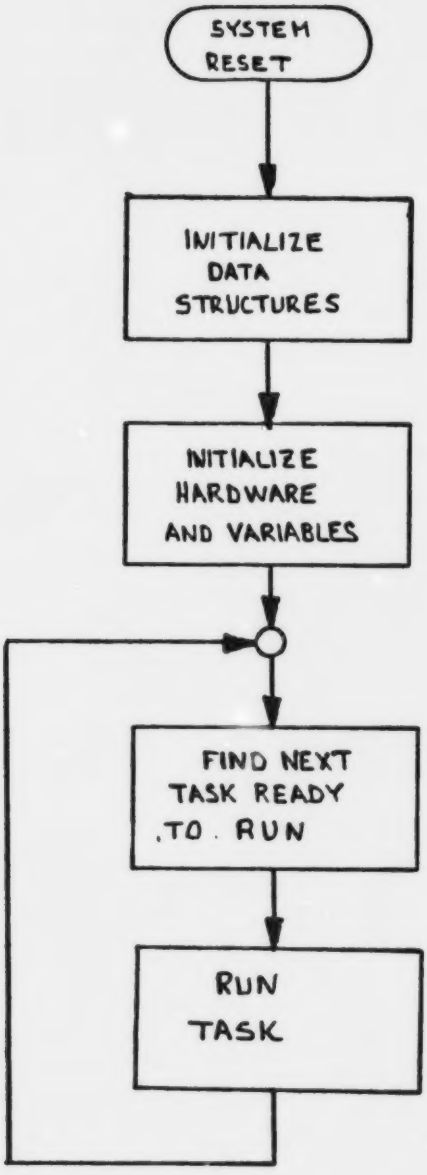


Figure 9. Initialization and Task Scheduling Module

Once the initialization process is complete, the microcomputer's interrupt system is enabled, which allows the real-time clock and control loop functions to begin executing. Next the module enters a loop which continually schedules the sequencer and control panel modules for execution. The sequencer is executed once per second and the control panel executes five times per second. The scheduler also blinks the CPU Status indicator lamp to indicate that the microcomputer is active and executing its program.

7.2 Sequencer Module

The sequencer module implements a finite state machine that sequences the charger through its functions. The state transition diagram for the finite state machine is shown in Figure 10.

The sequencer starts after the microcomputer is initialized. The first state after initialization carries out parameter initialization where the operator enters data required by the battery charger before charging can begin. This includes the following items:

a) present time of day and day of week
b) time when charging should begin
c) day of week for equalization charge
d) time when equalization charge should start
e) battery type
f) nominal battery voltage
g) maximum current which can be drawn from the AC line

Following parameter initialization, the charger enters the Idle state until the time to start charging or until the operator explicitly commands the charger to begin charging. Once one of these conditions occurs, a transition is made to the Constant Current Charge state. The contactor is closed and the current amplitude command is incremented until the charging current setpoint is reached. This setpoint is determined by the amount of current that can be drawn from the AC line. The charger stays in this state until the battery voltage has risen to the gassing voltage. At that point the sequencer makes a transition to the Constant Voltage Charge state.

In the Constant Voltage Charge state, the current amplitude command is adjusted to keep the battery voltage within a narrow band around the gassing voltage. The charging current decreases as the battery approaches full charge. When the charging current drops to five or six amperes, the sequencer makes a transition to one of two states.

If an equalization charge is not scheduled that day, the Finishing Charge state is entered. Otherwise, the Equalization

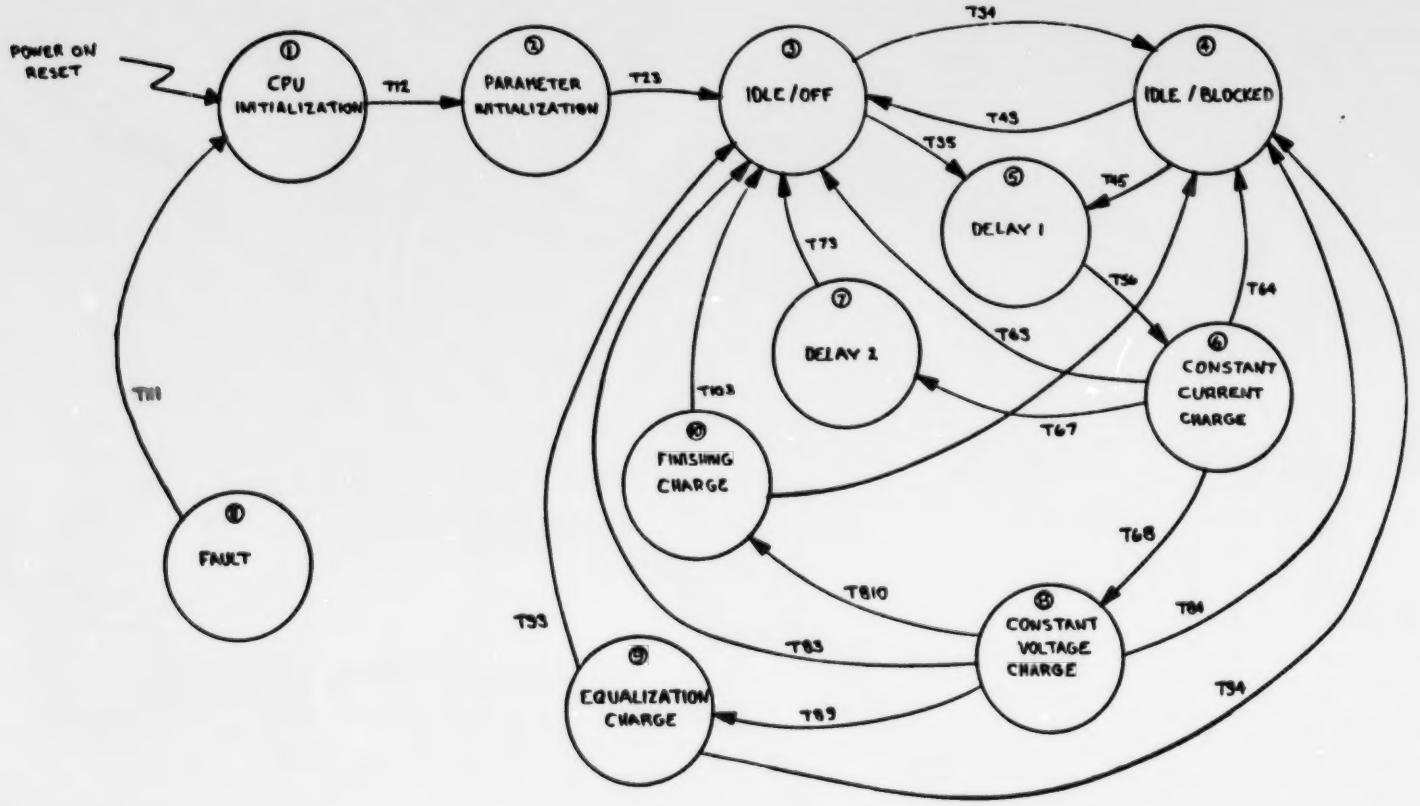


Figure 10. Battery Charger Sequencer State Transition Diagram

Charge state is entered. These two states differ only in the length of time that they take. The charging current is maintained at five to six amperes during these states. When the required time has elapsed, the sequencer makes a transition back to the Idle state. The battery is now charged.

There are two states which are only entered under abnormal circumstances. The Blocked state is entered if the charger is not connected to AC power or if the battery temperature is below zero Celsius. The charger is inactive in this state. The sequencer will return to the normal charging sequence if AC power is connected or if the battery temperature rises above zero. The Fault state is entered if one or more of the following conditions is detected:

a) battery voltage too high
b) battery voltage too low
c) battery temperature too high
d) excessively long charging time

Upon entering the Fault state, the charger is shut down and the Fault indicator lamp is lit. The Fault state can only be left by resetting the microcomputer.

7.3 Control Panel Module

This module handles the charger's interaction with the human operator via the portable terminal used as a control panel. The general flow of data is shown in Figure 11. The driver routine for the serial interface is contained in the Real-Time Clock module since it executes every 20 milliseconds. This guarantees that all characters sent from the terminal will be received.

The serial interface driver routine gets data to be sent to the terminal from an output circular buffer and places data received from the terminal into an input circular buffer. The control panel module gets commands from the terminal from the input circular buffer and sends messages to the terminal through the output circular buffer.

The control panel software is menu oriented. After leading the operator through the parameter initialization sequence, the control panel enters the set of menu sequences shown in Figure 12. These menus, and the commands used to move through the menus, are described in the Operator's Manual (Appendix A).

7.4 Real-Time Clock Module

The real-time clock software is activated by an interrupt (INT 7.5) generated by Counter #0 in the 8253 Counter/Timer IC. This counter counts a 250 kHz signal derived from a crystal oscillator on the microprocessor board. The counter is programmed to

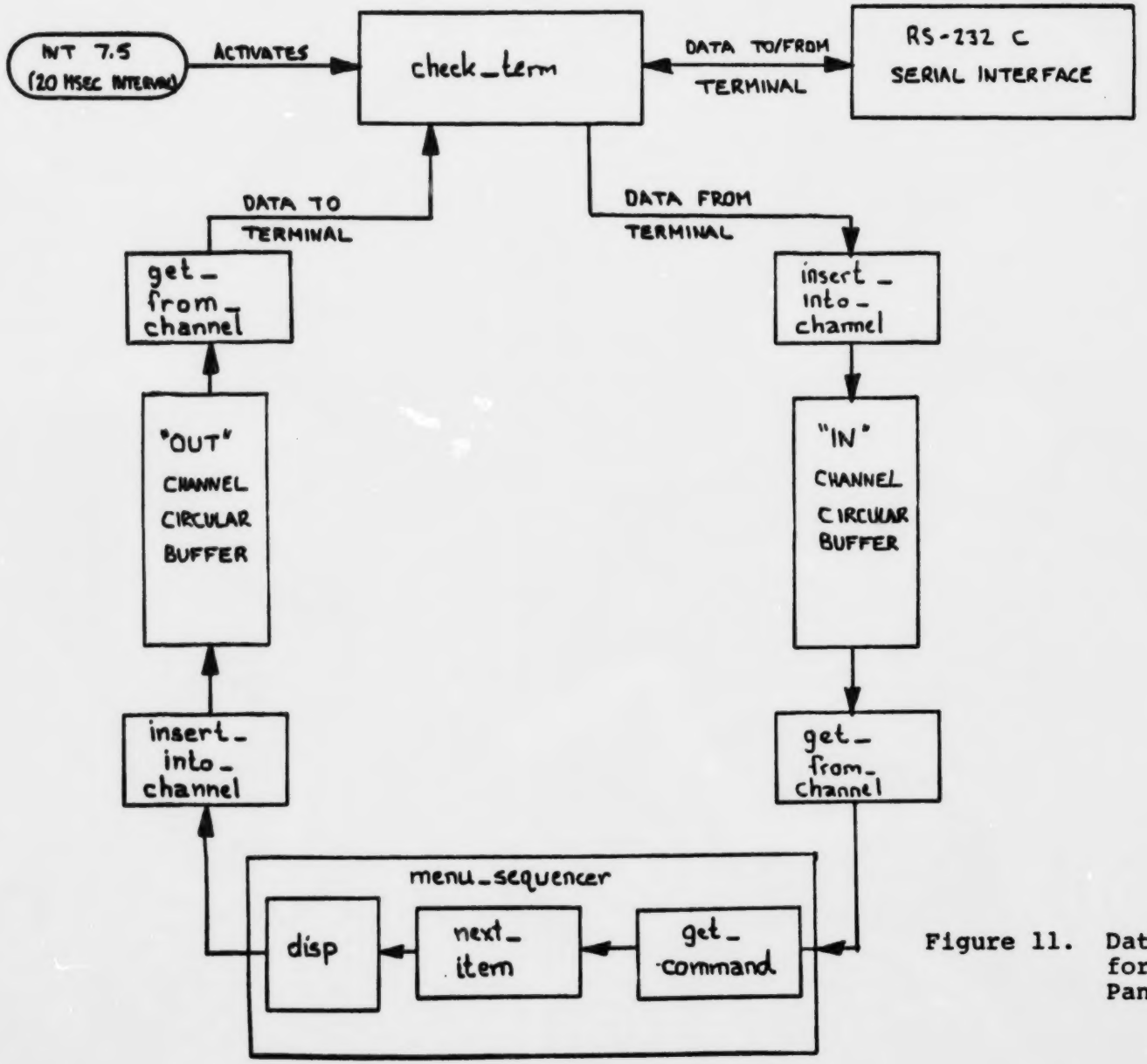


Figure 11. Data Flow for Control Panel Module

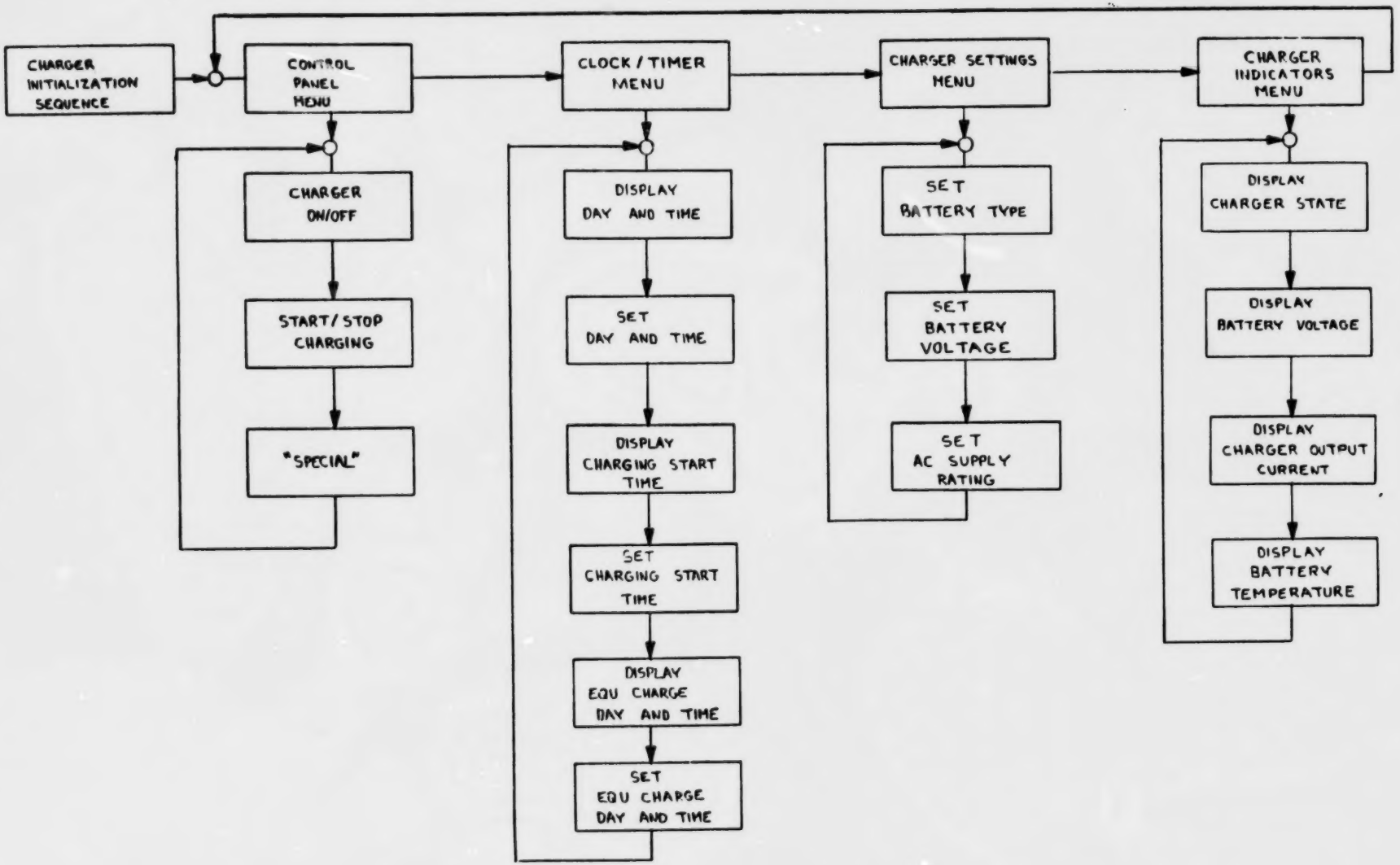


Figure 12 Control Panel Menu Sequences

generate an interrupt pulse every 20 milliseconds.

When the real-time clock software is activated, it updates the data structures which contain the present time of day and day of week. It also updates software timers used in scheduling the execution of the sequencer and control panel modules. If charging is in progress, a software timer which contains the elapsed time for the charging cycle is also updated.

7.5 Control Loop Module

Figure 13 shows a basic flowchart for the control loop module. Execution begins when Counter #1 in the 8253 Counter/Timer IC sends an interrupt signal (INT 6.5) to the microprocessor, signalling the end of an analog-to-digital conversion. The digitized sample of the selected feedback signal is retrieved form Counter #2 of the 8253 and is converted to the proper internal data format.

What happens next depends on which feedback signal has just been sampled and on which state the sequencer is in. If the signal just sampled is the charging current and the charger is in the Constant Current Charge state, the Finishing Charge state, or the Equalization Charge state, the new sample is compared to the charging current setpoint. If the actual charging current is outside a narrow band around the setpoint, the current amplitude command is adjusted to force the charging current towards the setpoint.

If the signal just sampled is the battery voltage and the charger is in the Constant Voltage Charge state, the new sample is compared to the gassing voltage for the battery. If the battery voltage is outside a narrow band around the gassing voltage, the current amplitude command is adjusted to force the battery voltage towards the gassing voltage.

No control loop action is taken if neither of the above conditions is met. The final action of the control loop module is to initiate another analog-to-digital conversion. The next feedback signal to be sampled is selected, Counter #2 starts counting the feedback pulses, and Counter #1 is programmed to generate an A/D-conversion-complete interrupt after 100 milliseconds.

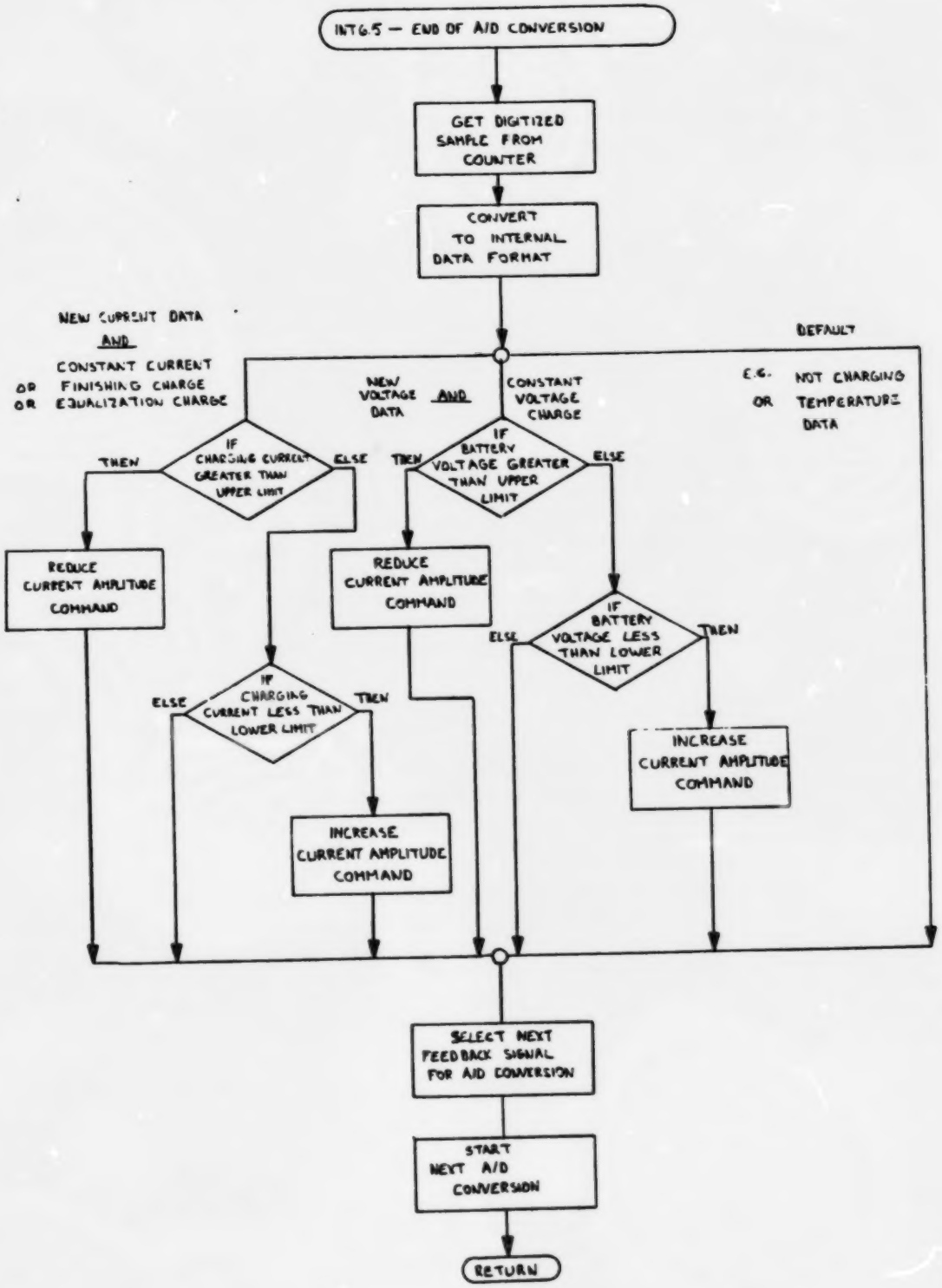

INT6.5 — END OF A/D CONVERSION
GET DIGITIZED SAMPLE FROM COUNTER
CONVERT TO INTERNAL DATA FORMAT
NEW CURRENT DATA AND CONSTANT CURRENT OR FINISHING CHARGE OR EQUALIZATION CHARGE
NEW VOLTAGE DATA AND CONSTANT VOLTAGE CHARGE
DEFAULT
E.G. NOT CHARGING OR TEMPERATURE DATA
IF CHARGING CURRENT GREATER THAN UPPER LIMIT
THEN
ELSE
IF BATTERY VOLTAGE GREATER THAN UPPER LIMIT
THEN
ELSE
REDUCE CURRENT AMPLITUDE COMMAND
REDUCE CURRENT AMPLITUDE COMMAND
IF BATTERY VOLTAGE LESS THAN LOWER LIMIT
ELSE
THEN
IF CHARGING CURRENT LESS THAN LOWER LIMIT
ELSE
THEN
INCREASE CURRENT AMPLITUDE COMMAND
INCREASE CURRENT AMPLITUDE COMMAND
SELECT NEXT FEEDBACK SIGNAL FOR A/D CONVERSION
START NEXT A/D CONVERSION
RETURN

Figure 13. Control Loop Module

8.0 AUXILIARY DC/DC CONVERTER

The auxiliary DC/DC converter is designed to charge an electric vehicle's 12 volt auxiliary battery from the propulsion battery. It is housed in a separate 18.7 cm x 18.7 cm x 6.4 cm aluminum case.

At present the converter is designed to operate from a 96 volt (nominal) battery back (i.e. 16 6 volt batteries). Other battery pack voltages in the range from 72 to 150 volts can be accomodated by changing the turns ratios of the transformers in the converter. Operation with lower voltage battery packs (e.g. 48 volts) will require some redesign in order to accomodate the higher currents.

Th operating envelope of the converter's output is shown in Figure 14. The unit will deliver up to 30 amperes to the 12 volt battery and its load as long as the output voltage is between 8 and 14.4 volts. Voltages below 8 volts indicate that the battery is defective, shorted, or disconnected. The battery gassing voltage is approximately 14.4 volts so the converter limits its maximum output voltage to this level.

Since the converter can deliver up to 30 amperes to the load, the auxiliary battery only has to deliver any current in excess of 30 amperes. In many cases this will allow the use of a small motorcycle type auxiliary battery. In that case the total size of the battery plus converter will be no greater than that of a conventional automobile battery.

The basic specifications of the converter are summarized below.

Maximum Output Voltage: 14.4 volts DC

Output Current Limit: 30 amperes

Input Voltage Range: 85 volts DC to 125 volts DC
Unit will operate down to 70 volts with some droop in the output voltage. Unit shuts down if input voltage exceeds 130 volts DC.

Efficiency: Better than 80% at full load

Ambient Operating Temperature: -20 C to +40 C
Unit shuts down when internal temperature exceeds 70 C.

Unit is protected against:

a) Reverse polarity connections on input and output
b) Output short circuit
c) Input overvoltage and transients
d) Overtemperature

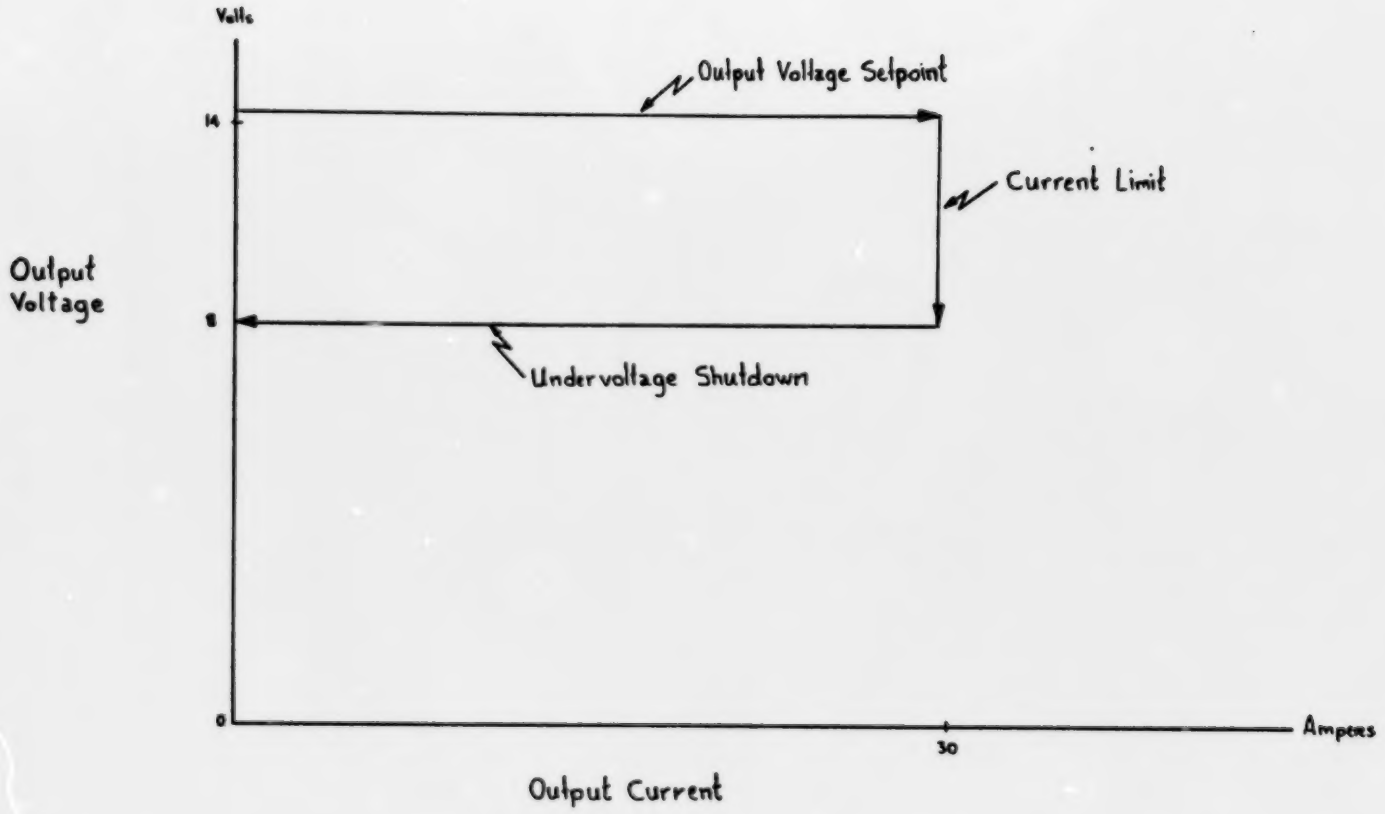


Figure 14. DC/DC Converter Operating Envelope

9.0 CONCLUSIONS AND RECOMMENDATIONS

The basic objective of producing a battery charger suitable for use on-board an electric vehicle has been met. The charging profile used follows the recommendations of battery manufacturers for modern lead-acid traction batteries. Since the charger is controlled by a microcomputer, it will be relatively easy to change the charging profile if a different type of battery is used.

The charger operates from either 120 VAC or 240 VAC single-phase household circuits. Depending on the power capacity of the circuit, the charger should be able to recharge an 80% discharged 16 kW-hr battery in five to twelve hours. The energy efficiency of the charger over the charging cycle has not been measured since a suitable set of batteries was not available. However, simulations of the charging cycle using measured values for the charger efficiency at different power levels indicate that the energy efficiency will meet or exceed the 85% goal.

There are some aspects of the battery charger's design that need further work. The charger is small enough to install on-board an electric vehicle. However, if the unit were smaller, it would be considerably easier to find space for it on the vehicle. The size could be reduced in the following ways:

1) Rearrange the components presently used in the charger to to increase the packaging density.

2) Consolidate the control circuits on to one circuit board, thereby eliminating the STD bus card cage.

3) Reduce the size of the transistor base drives and their associated power supplies by consolidating them on to one or two printed circuit boards and by increasing the density of components on these boards.

4) Increase the operating frequency of the power conversion section. This would decrease the size of the transformers and of filter components but would entail a substantial design effort.

5) Decrease the peak power output of the charger.

The cost of the charger is currently too high to make it commercially viable. In part this is because the charger is still in the engineering prototype stage. In preparing a production prototype, the following steps could be taken to reduce manufacturing costs:

1) Redesign the control circuit using a single chip microcomputer and other highly integrated components so that the control circuit fits on to a single printed circuit board.

2) Simplify the transistor base drive circuits and base drive power supplies to reduce the number of components required.

3) Design a control panel using an inexpensive membrane keypad and a small LCD (liquid crystal) alphanumeric display to replace the hand-held terminal used in the prototype.

4) Find alternative sources for expensive components such as high frequency filter capacitors and ferrite transformer cores.

Even with these changes, it is likely that the charger would still have a direct manufacturing cost approaching $1000. The cost will decline somewhat in the future since electronic components have a declining cost trend. Unfortunately, components such as capacitors and transformers will remain expensive. The long term solution to reducing costs is probably one of the following:

1) Redesign the charger with low cost being the primary goal. This will require sacrificing some of the desirable properties of the current design.

2) Manufacture the charger in high volumes. Many of the components in the charger are expensive because they are intended for industrial rather than consumer use and thus are produced in relatively small quantities. If they were purchased in large volumes, the price would drop substantially. Volume production would also reduce labor costs and overhead.

3) Integrate the charger into the vehicle drive system. The battery charger and the chopper or inverter used to control the vehicle's motor contain many similar components and circuits. By combining the functions of the charger and the motor controller, it may be possible to share some of these components and circuits, thus reducing the effective cost of the battery charger.

REFERENCES

1. S. Gross, "Rapid Charging of Lead-Acid Batteries", Conference Record of IEEE IAS Annual Meeting, Milwaukee Wisconsin, October 1973, pp. 905 - 912

2. R.H. Sparks, "Rapid Charging Batteries for Electric Propulsion Systems", SAE Automotive Engineering Congress, Detroit Michigan, January 1972, SAE Paper No. 72019.

3. J.J. Smithrick, "Pulse Charging of Lead-Acid Traction Cells", NASA Lewis Research Centre, Cleveland Ohio, May 1980, DOE/NASA/1044-6.

4. W.G. Sunu and B.W. Burrows, "Electrolyte Stratification and Charge Characteristics of Industrial Lead-Acid Cells", Power Sources - 8, Academic Press, London, 1980.

5. J.L. Weininger and F.G. Siwek, "A System Evaluation of Lead-Acid Battery Chargers", Fourth International Electric Vehicle Symposium, Dusseldorf, September 1976.

6. J.J. Smithrick, "Rapid, Efficient Charging of Lead-Acid and Nickel-Zinc Traction Cells", Intersociety Energy Conversion Conference, 1978, pp. 672 - 676, SAE Paper No. 789122.

7. J.J. Rowlette, "Modified Coulometer State-of-Charge Determination for Lead-Acid Batteries", Electric Vehicle Council Symposium VI, Baltimore Maryland, October 1981, EVC No. 8164.

8. J.L. Weininger and F.G. Siwek, "A System Evaluation of Lead-Acid Battery Chargers", Fourth International Electric Vehicle Symposium, Dusseldorf, September 1976.

9. J.L. Van Eck, G. Maggetto, P. Mathys, A. Broucke, "Battery Chargers for Commercial Electric Vehicles. A Comparison of Different Solutions", Electric Vehicle Council Symposium VI, Baltimore Maryland, October 1981, EVC No. 8163.

10. D. Gosden, "An Electric Vehicle Battery Charger", Electrical Engineering Transactions, Institute of Engineers, Australia, 1978, Vol. EE-14, pp. 88 - 91.

11. W.E. Rippel, "Optimizing Boost Chopper Charger Design", Proceedings of the Sixth National Solid-State Power Conversion Conference (Powercon 6), May 1979.

12. T.S. Latos and D.J. Bosack, "A High Efficiency 3 kW Switchmode Battery Charger", Power Electronics Specialists Conference (PESC), June 1982.

13. G. Chryssis, High Frequency Switching Power Supplies: Theory and Design, McGraw-Hill Book Co., New York, 1984.

14. W. Rippel and W.T. McLyman, "Design Techniques for Minimizing the Parasitic Capacitance and Leakage Inductance of Switched-Mode Power Transformers", Proceedings of the Ninth International Solid-State Power Electronics Conference (Powercon 9), Washington D.C., July 1982.

15. E.C. Whitcomb, "Designing Non-Dissipative Current Snubbers for Switched-Mode Converters", Proceedings of the Sixth International Solid-State Power Electronics Conference (Powercon 6), May 1979.

16. S.P. Hsu, A. Brown, L.Rensink, and R.D. Middlebrook, "Modelling and Analysis of Switching DC-to-DC Converters in Constant-Frequency Current Programmed Mode", Proceedings of the IEEE Power Electronics Specialists Conference, San Diego CA, June 1979.

17. ISB-3111 8085 Based Processor Card: User's Preliminary Information, Intersil, Inc., Systems Division, Sunnyvale CA, 1982.

18. MDX EPROM/UART: Operation Manual, Mostek Corporation, Carrollton TX, February 1981, Publication No. MK 79604.

19. MCS-96 User's Manual, Intel Corp., Santa Clara CA, 1984.

20. B.W. Kernighan and D. Ritchie, The C Programming Language, Prentice-Hall, Englewood-Cliffs NJ, 1978.

APPENDIX A INSTALLATION AND OPERATION MANUAL

This manual covers installation and operation of the prototype battery chargers. Before installing the charger, the installer should have read the sections in this manual dealing with installation of the charger and with the start-up procedure for the charger.

A.1 Installation

Installation of the battery charger must be performed only by technically qualified personnel familiar with the safety procedures required for working with electrical circuits and lead-acid batteries.

A.1.1 Physical Installation

The charger must be installed in a location that is not subject to liquid splashing or spraying or to significant quantities of airborne particles. The ambient air temperature around the charger must not be allowed to exceed 45 Celsius while it is operating.

Allow a clearance of at least 10 cm at the two ends of the charger containing the ventilation openings. This clearance is required for ventilation and for the connectors plugged into the charger. The charger should be mounted so that the indicator lamps and reset pushbutton are both visible and accessible.

A.1.2 Electrical Installation

a) Control Panel

The prototype charger can use any ASCII data terminal that communicates at 300 baud over a standard RS-232C serial link as a control panel. For on-board use this could be a hand-held computer such as a Radio Shack PC-2 (with RS-232 adapter), a lap-top computer such as a Tandy Model 100 or NEC 8201, or a hand-held terminal such as the G.R. Electronics Pocket ASCII Terminal. For laboratory testing, a standard video display terminal, such as a DEC VT-100, Hazeltine Esprit, Qume 102G, Televideo 950, etc., will be more convenient.

The serial interface in the battery charger has the following characteristics:

Interface Type	RS-232C DCE (Data Communications Equipment) Full duplex
Connector	Female DB-25

Data Rate	300 baud
Data Format	7 data bits, one stop bit, even parity

If the terminal serial interface is configured as DTE (Data Terminal Equipment), which is usually the case, a standard cable with male DB-25 connectors on either end may be used. The cable should support the following signals:

Signal	Connector Pin Number
Chassis Ground	1
Transmit Data	2
Receive Data	3
Request to Send	4
Clear to Send	5
Data Set Ready	6
Signal Ground	7
Data Terminal Ready	20

If the terminal interface is configured as DCE, a null modem connection must be used.

b) Battery Power

The vehicle propulsion battery which is connected to the output of the charger must be a lead-acid battery with a minimum of 45 series connected cells (15 six volt traction batteries in series) and a maximum of 54 series connected cells (18 six volt traction batteries in series). At a nominal cell voltage of 2 volts, this corresponds to a range of 90 V to 108 V. In order for the charger to start properly, the battery open-circuit voltage must not be less than 75 volts.

Connect the battery to the charger output terminals with insulated copper cables with a conductor size of AWG 10 or heavier. Connecting lugs must be rated to carry 50 amperes. Make the most direct connection possible. Do not insert switches or other components in the circuit. Pay careful attention to the polarities marked on the output terminal block.

The charger draws 200 to 300 milliamperes whenever it is connected to the battery so there may be some small sparks as the connections are made to the charger. It may be advisable to make the battery connections first so that the sparks occur at the charger rather than at the battery.

The connections to the battery charger should be covered with an insulating shroud or boot. This may be attached to the terminal block which holds the output terminal studs.

c) Battery Temperature Sensor

The battery temperature sensor is a small solid-state device (National Semiconductor LM35) encased in plastic. It is attached to a cable which has a 3-pin XLR type plug at the other end. This plug is connected to the corresponding receptacle on the battery charger. An extension cable can be made with three-conductor cable, a female XLR connector (e.g. Cannon XLR-3-11C), and a male XLR connector (e.g. Cannon XLR-3-12C).

The sensor must be mounted on the exterior of the battery in a location where the temperature closely approximates the internal electrolyte temperature. The sensor is not resistant to battery acid, so it must not be immersed in the electrolyte. A battery terminal post is one possible location since it is directly connected to a metal grid immersed in the electrolyte. The side of the battery case is also a possible location if the battery case is relatively thin. The accuracy of the temperature sensor can be improved by placing thermal insulation over the sensor once it is mounted on the battery. This diminishes the effect of the ambient air temperature on the sensor's output.

d) AC Power

DO NOT CONNECT THE CHARGER TO AC POWER BEFORE THE CHARGER HAS BEEN CONNECTED TO THE VEHICLE BATTERY AND THE CHARGER'S CONTROLLER HAS BEEN INITIALIZED.

The charger may be connected to single phase AC lines with the following ratings:

110 to 125 VAC	15 to 30 amperes
208 to 240 VAC	15 to 30 amperes

Use a three conductor cable with a voltage rating of at least 250 VAC to connect the charger to the AC line. Conductors must be AWG 16 or heavier if operating from a 15 ampere circuit. Conductors must be AWG 14 or heavier if operating from a 30 ampere circuit.

The connector that plugs into the AC receptacle on the battery charger is an EEMAC L6-30P type (e.g., Hubbell 2623). The connector on the AC line side should be chosen to match the receptacle. Make sure that the ground pins on the connectors are connected and that a grounded AC line receptacle is used.

The ground connection for the L6-30P connector is shown below.

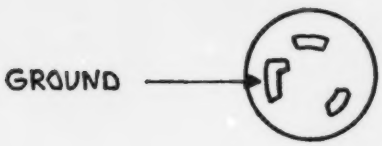


Figure 15 L6-30P Connector

A.2 Charger Operation

A.2.1 Start-up Procedure

The start-up procedure must be performed every time the charger is connected to a new battery, or the internal microcomputer is reset. Connect the data terminal (control panel), battery, and battery temperature sensor to the charger as described in the installation instructions. Check the CPU status indicator lamp. It should be blinking regularly, approximately once or twice a second. If it is not blinking, or is blinking erratically, push the red CPU reset button and then release it. Wait 20 to 30 seconds. If the status light does not start blinking properly, repeat the procedure.

Once the microcomputer has been reset, check the terminal. It should be repeating the following message:

INITIALIZE CHARGER

If you don't see this message, try resetting the microcomputer once more. If that has no effect, check your terminal to ensure that it is set up for the data format and data rate specified in the installation instructions. If you are using a pocket or lap-top computer, make sure that it is in terminal mode.

Once your terminal is receiving the INITIALIZE CHARGER message, enter the following on the terminal's keyboard:

YES <CR>

<CR> indicates the "carriage return" or "enter" key on the keyboard. This starts the charger initialization sequence.

The charger should respond with:

SET PRESENT TIME - DAY:

If the INITIALIZE CHARGER message continues to appear, enter

the YES<CR> command again until the charger pays attention.

Once the terminal displays the SET PRESENT TIME - DAY: message, enter the current day of the week. For instance

TUESDAY<CR>

If you misspell the day of week or enter invalid data, the charger will respond with

DAY OF WEEK?

and wait for a valid entry. Otherwise the charger responds with

HOUR (24 HR CLOCK)?

Enter the present hour of the day using a 24 hour clock. For example, if the present time is 2:35 PM, enter

14<CR>

The charger will respond with

MINUTES?

Enter the number of minutes past the present hour. For example, if the present time is 2:35 PM, enter

35<CR>

The charger will respond with

CHARGING START TIME - HOUR?

Decide on the time of day when you wish charging to commence and enter the hour using a 24 hour clock. For example

22<CR>

The charger will respond with

MINUTES?

Enter the number of minutes after the hour, for example

30<CR>

The charger will respond with

EQUALIZATION DAY?

Enter the day of the week when an equalization charge should

begin, for example

SATURDAY<CR>

The charger will respond with

EQUALIZATION START TIME - HR?

Enter the hour of day (using a 24 hour clock) that the equalization charge should start. Equalization usually takes two hours longer than a regular charge.

The charger will respond with

EQU START TIME - MINUTES?

Enter the number of minutes after the hour for the equalization charge starting time.

The next message from the charger is

BATTERY MENU (PRESS ENTER KEY)

Press the "carriage return" or "enter" key to continue. You will be presented with a choice of battery types, select the one that matches the battery connected to the charger. The menu entries are

EXIDE EV-106 (Y/N)?
EXIDE XPV-23 (Y/N)?
GLOBE UNION EV1000 (Y/N)?
GENERIC LEAD-ACID (Y/N)?

To select an entry, enter Y<CR>. To move on to the next entry, enter N<CR>. The GENERIC entry should be selected if your battery isn't one of the three types listed.

The next message from the charger is

NOMINAL BATTERY VOLTAGE?

This is the voltage of the battery, assuming a cell potential of 2 volts. Only four responses are valid:

90<CR>
96<CR>
102<CR>
108<CR>

The final query in the initialization sequence is

AC SUPPLY RATING (AMPS)?

Enter the maximum current which may be drawn by the charger from the AC supply circuit. This must be a whole number between 6 and 25. The charger will not accept entries of less than 6 amperes and will truncate entries greater than 25 amperes to 25 amperes.

The charger has now been initialized. It will display

CHARGER CONTROL PANEL MENU

indicating that it is in the upper level menu sequence. Any of the values entered during the initialization sequence can be changed by selecting the appropriate menu item and entering the new value.

Now that initialization is complete, the charger may be connected to the AC line.

A.2.2 Control Panel

The control panel software is menu driven. Figure 16 shows the menus available to the operator. The operator moves through the menus and makes selections of actions to be performed by entering commands and data at the keyboard. The commands recognized by the control panel software are:

a) Y<CR> (Yes)	Selects the current menu item. This will cause either a descent to a lower level menu or execution of the action indicated.
b) N<CR> (No)	Ignores the current menu item and moves on to the next menu item as indicated by the arrows in Figure 16.
c) F<CR> (Forward)	Ignores the current menu item and moves on to the next menu item as indicated by the arrows in Figure 16.
d) B<CR> (Backward)	Ignores the current menu item and moves to the previous menu item as indicated by the arrows in Figure 16.
e) X<CR> (eXit)	Ignores the current menu item and moves up to a higher level menu.

Figure 17 shows how these commands are used. For instance, suppose that the battery charger has just been initialized and the terminal is displaying the CONTROL PANEL MENU message. The operator wishes to change the nominal battery voltage because it was entered incorrectly during initialization. The operator moves to the CHARGER SETTINGS MENU by entering F<CR> twice. Then

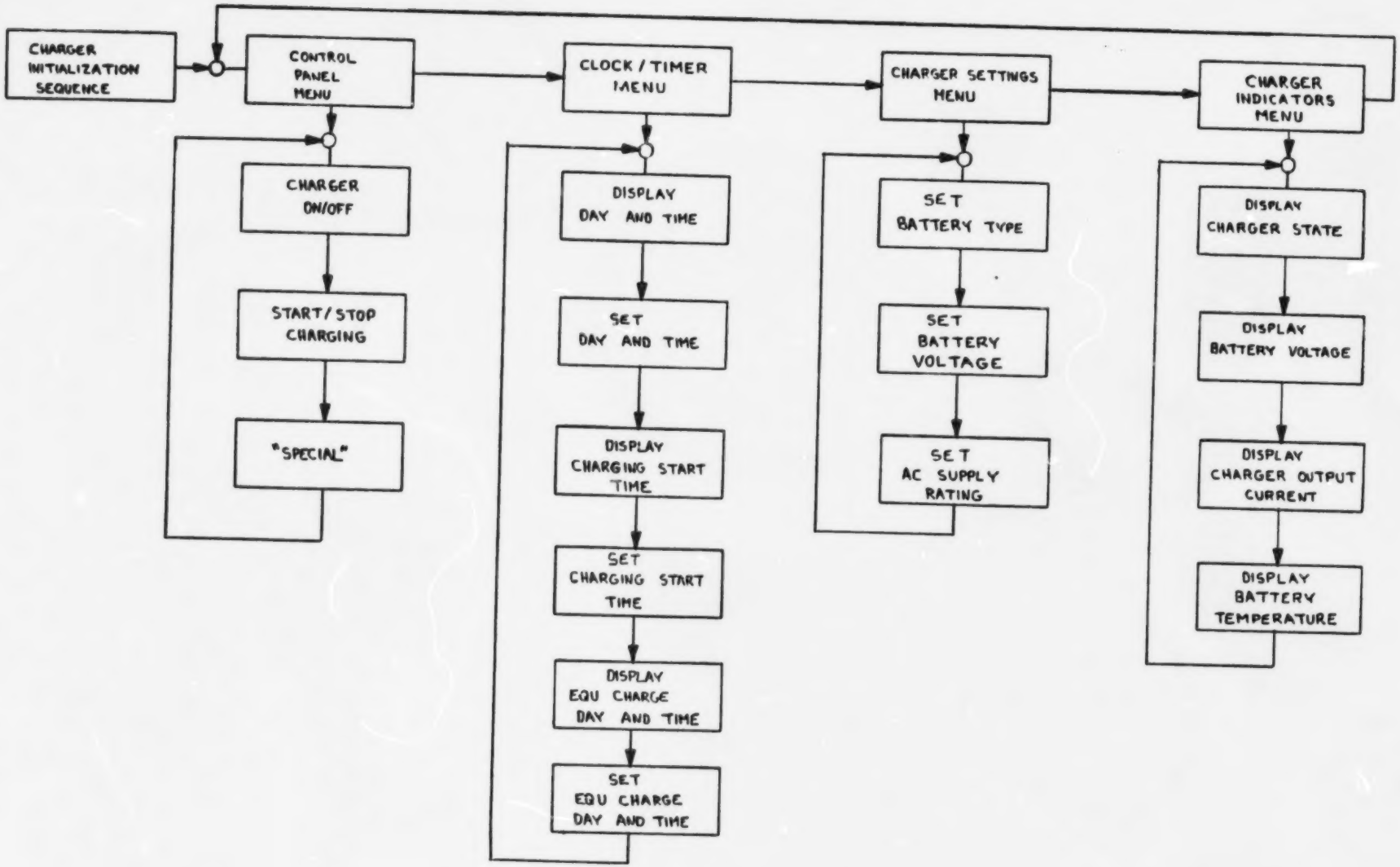


Figure 16. Control Panel Menus

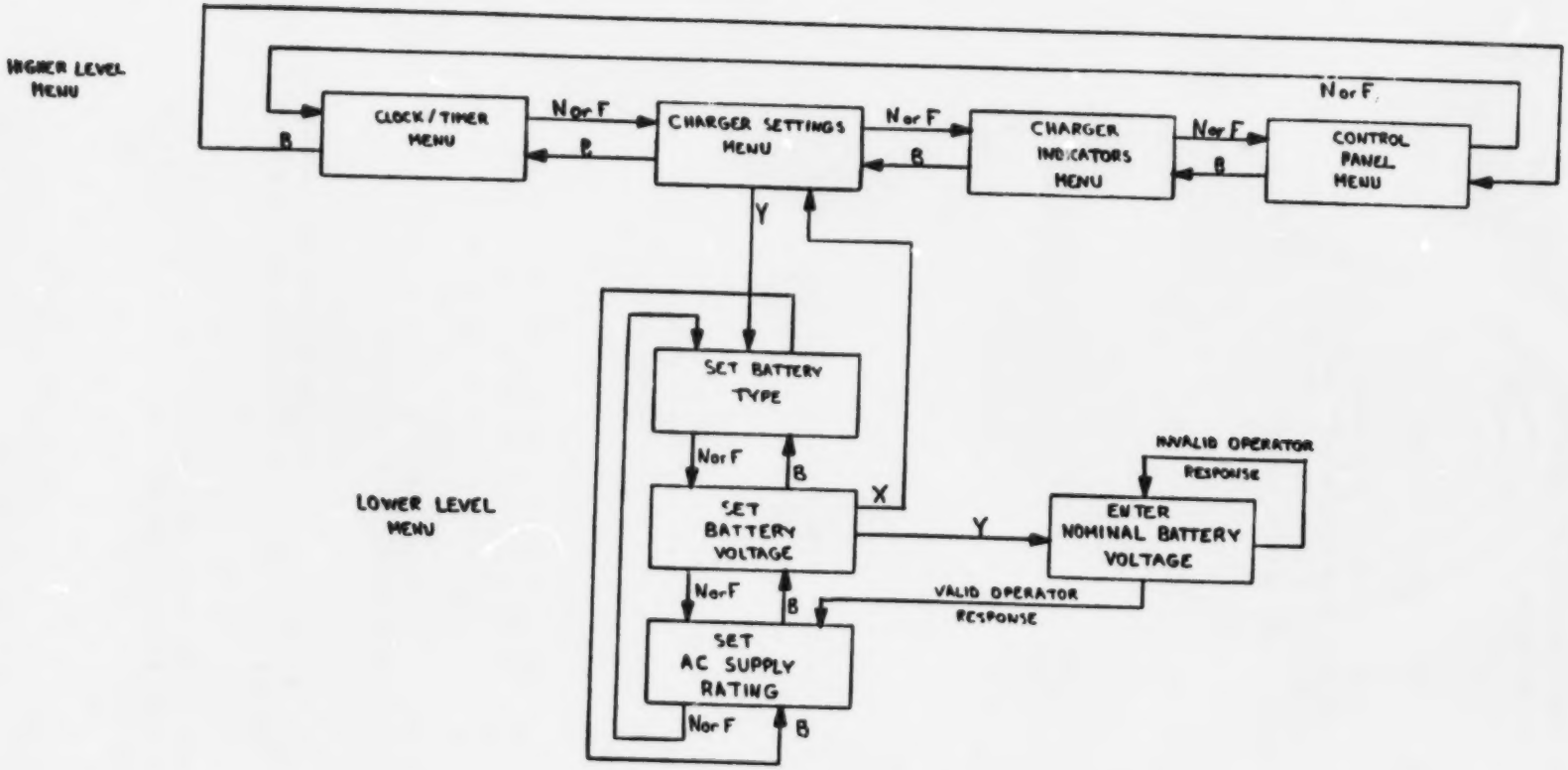


Figure 17. Use of Control Panel Commands

he or she enters Y<CR> to select this menu. This causes the first item in the menu (SET BATTERY TYPE) to be displayed. The operator enters N<CR> or F<CR> to move to the desired item. Then Y<CR> is entered to select the item. The charger now displays a prompt message:

ENTER NOMINAL BATTERY VOLTAGE

The operator enters the correct value, for example 96<CR>. This valid response causes the control panel to go to the next item in the menu (SET AC SUPPLY RATING). To escape from the menu, the operator now enters X<CR>, causing the control panel to return to the main menu.

A.2.3 Control Panel Menus

The main menu allows the operator to select four lower level menus:

a) Control Panel Menu

b) Clock/Timer Menu

c) Charger Settings Menu

d) Charger Indicators Menu

Control Panel Menu The control panel menu gives the operator direct control over charger operation. The items in this menu are:

1) Turn charger on/off This item has a toggle action when it is selected. If the charger is on, the charger is turned off. If the charger is off, it is turned on. When the charger is off, the microcomputer is still active but charging is disabled.

2) Start/Stop charging This item also has a toggle action when it is selected. If charging is not in progress, charging is started. If charging is in progress, charging is stopped. This allows the operator to start charging at times other than the preset starting time and allows the operator to terminate charging at any point in the charging cycle.

3) Special This item is reserved for diagnostic and development purposes. The operator should never select this item.

Clock/Timer Menu The clock/timer menu allows the operator to read and control the charger's internal clock. The items in this menu are:

1) Display Present Time Displays the present time of day and day of week. A 24 hour clock is used.

2) Set Present Time Allows the operator to set the present time of day and day of week. A 24 hour clock must be used.

3) Display Charge Time Displays the time of day when charging will commence.

4) Set Charge Time Allows the operator to set the time of day when charging will commence. A 24 hour clock must be used.

5) Display Equalization Time Displays the time of day and day of week when the equalization charge will commence.

6) Set Equalization Time Allows the operator to set the day of week and time of day when the equalization charge will start. A 24 hour clock must be used.

Charger Settings Menu The charger settings menu allows the operator to change the parameters entered into the charger during charger initialization. The menu items are:

1) Set Battery Type Activates a sub-menu of four different battery types. The operator must select the type which most closely matches the battery attached to the charger. This information is used by the charger to determine which gassing voltage equation it will use.

2) Set Nominal Battery Voltage The operator must enter the nominal voltage (assuming 2 volts per cell) of the battery attached to the charger. Valid entries are 90, 96, 102, and 108.

3) Set AC Supply Rating The operator must enter the maximum current which can be supplied to the charger by the AC line. This must be a whole number between 6 and 25.

Charger Indicators Menu The charger indicators menu acts as the indicator lamps and meters for the charger. The menu items are:

1) Display Charger State When this item is selected, the current status of the battery charger is displayed on the terminal. The status messages are:

IDLE/OFF The charger is not charging. It is

either waiting to begin charging or it is off.

CONSTANT CURRENT CHARGE The charger is charging and is in the initial constant current charge portion of the charge profile.

CONSTANT VOLTAGE CHARGE The charger is charging and is in the constant voltage portion of the charge profile.

FINISHING CHARGE The charger is charging and is in the final low current portion of the charge profile.

EQUALIZING CHARGE The charger is charging and is in the extended low current portion of the charge profile used when equalizing the battery.

BLOCKED - LOW TEMPERATURE The charger cannot begin charging because the battery temperature is below zero Celsius. The battery should be heated so that its temperature is above zero Celsius. Charging a battery with an electrolyte temperature below zero Celsius is very inefficient.

BLOCKED - NO POWER The charger cannot begin charging because no AC power is available. The operator should ensure that the charger is connected to the AC line and that the AC line is energized.

FAULT - BATTERY OVERTEMP The charger has shut down because the battery was too hot. The battery should be checked for damage. The charger must be reset with the reset pushbutton and reinitialized.

FAULT - BATTERY OVERVOLTS The charger has shut down because the battery voltage was too high. The battery should be checked for damage or poor connections to the charger. The charger must be reset with the reset pushbutton and reinitialized.

FAULT - BATTERY LOWVOLTS The charger has shut down because the battery voltage was too low. The battery should be checked for shorted cells or other damage. The charger must be reset with the reset pushbutton and

reinitialized.

FAULT - OVERTIME The charger has shut down because charging was taking an excessive amount of time. The battery should be checked for damage and for sneak discharge paths. The charger must be reset with the reset pushbutton and reinitialized.

2) Display Battery Voltage The battery voltage is displayed on the terminal.

3) Display Charging Current The charger output current is displayed on the terminal.

4) Display Battery Temperature The battery temperature, as measured by the temperature sensor, is displayed on the terminal.

A.2.4 Operating Guidelines

The charger is intended for vehicles in commuter or delivery applications which are used daily. The charger will recharge the battery every day, starting at a preprogrammed time. Normally, the charger will be connected to the AC supply in the evening when the vehicle returns to its garage and charging will start some time later.

The operator should select a charging start time in the evening that allows the charger to completely charge the battery by the time the vehicle is required the next morning. The time required for charging will depend on the following factors:

a) Depth of discharge of battery
b) Energy storage capacity of battery
c) Power capacity of AC supply

If the battery is heavily discharged (i.e. more than 70%) and a 120 V, 15 A AC supply is used, the operator should allow at least 10 hours for charging.

If there is an opportunity for a partial or total recharge during the day, the operator can begin charging using the START CHARGING command and can end charging using the STOP CHARGING command.

The charger automatically performs an equalization charge once per week, starting at a preprogrammed time. Since the equalization charge typically takes two hours longer than a regular charge, the operator should select a day for the equalization charge where charging can start earlier or end later.

There may be situations where automatic recharging at the programmed time is not required. The operator may disable recharging with the TURN CHARGER OFF command. Recharging is reenabled with the TURN CHARGER ON command. Automatic recharging should not be disabled by disconnecting the charger from the AC line. If this is done, the charger will begin charging immediately if the charger is reconnected after the programmed starting time.

There may also be situations where the vehicle can not be connected to the AC supply before the programmed starting time. In that case the operator should use the STOP CHARGING command to return the charger to the IDLE state before connecting the charger to the AC supply. Charging may then be initiated by the START CHARGING command.

If the charger must be disconnected from the AC supply while charging is still in progress, the STOP CHARGING command should be used first to stop charging. The charger may then be disconnected. When the charger is reconnected to the AC supply, the charger will resume charging at the next programmed starting time unless the START CHARGING command is used to manually initiate charging.

APPENDIX B

SCHEMATIC DIAGRAMS

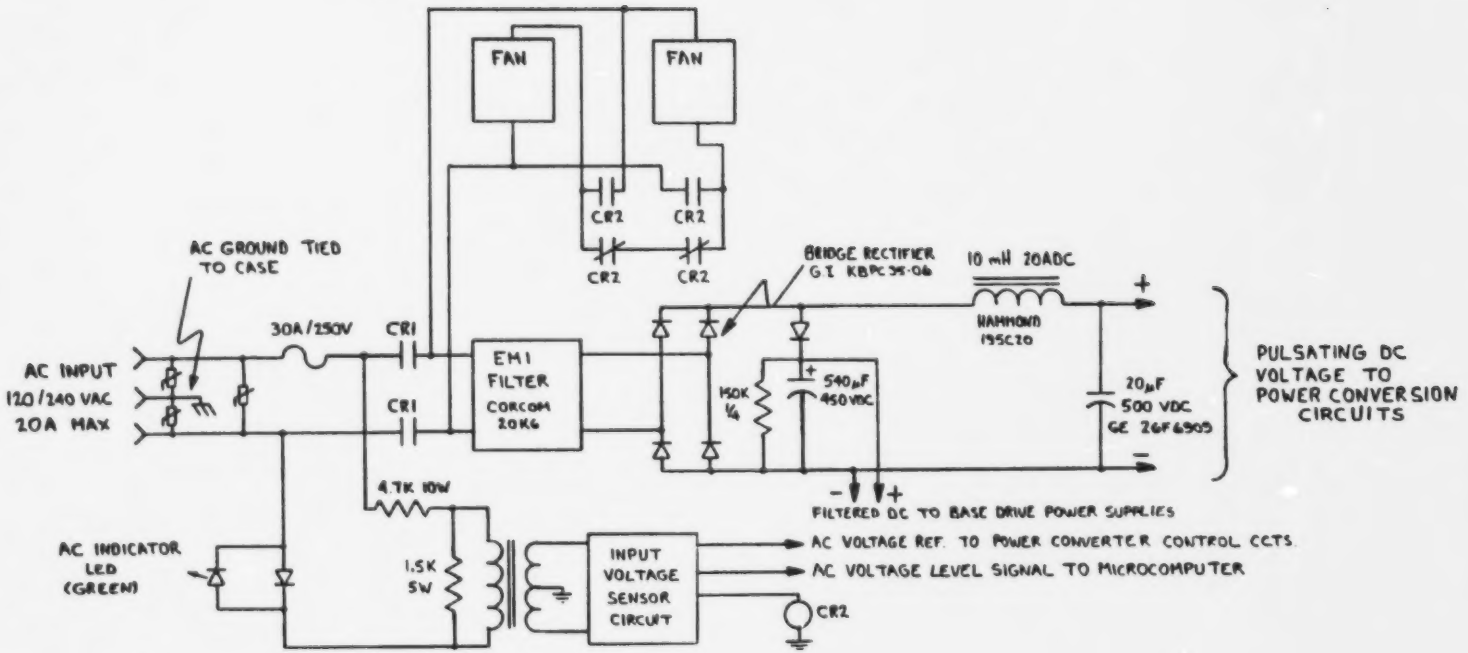
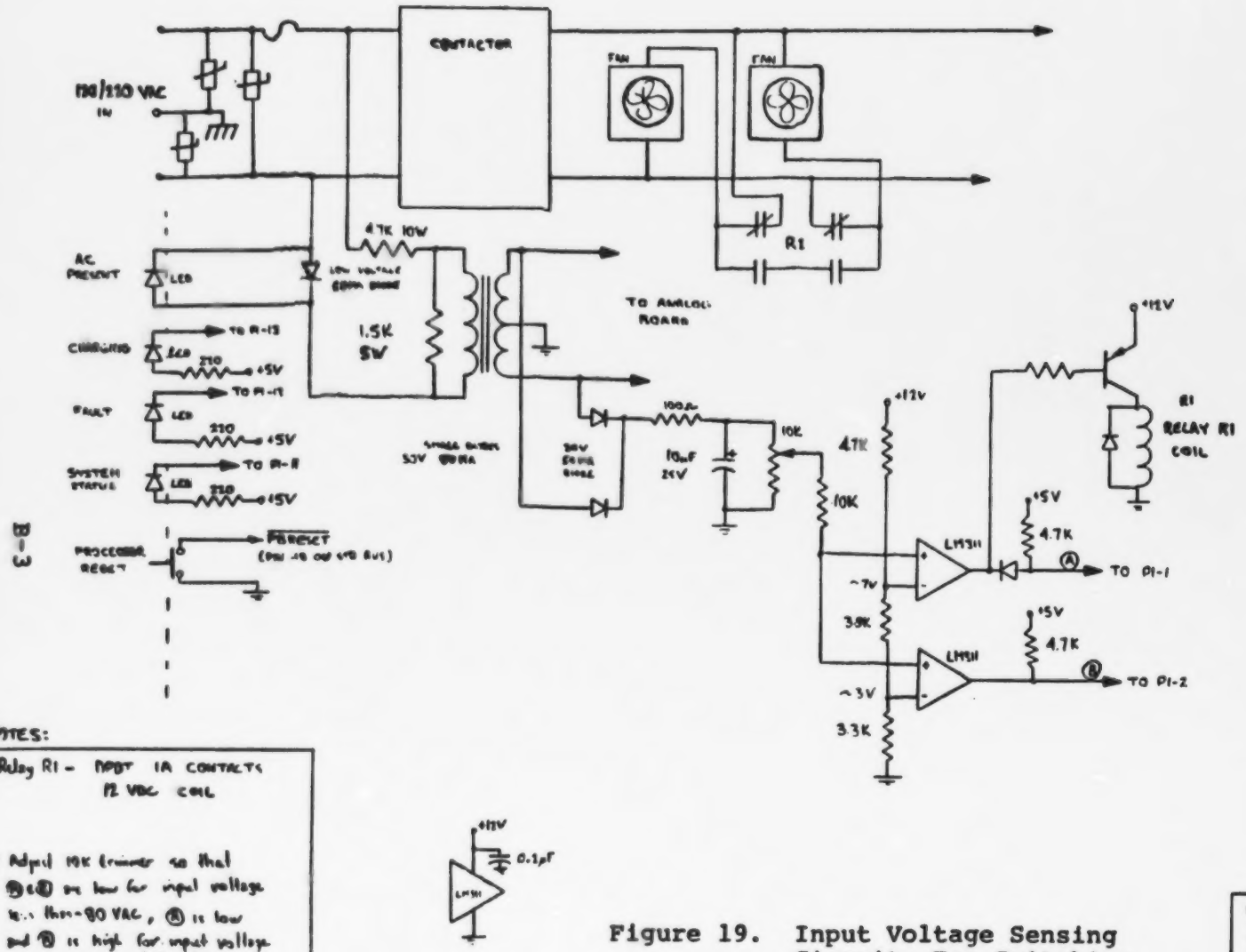


Figure 18. Charger Input Section



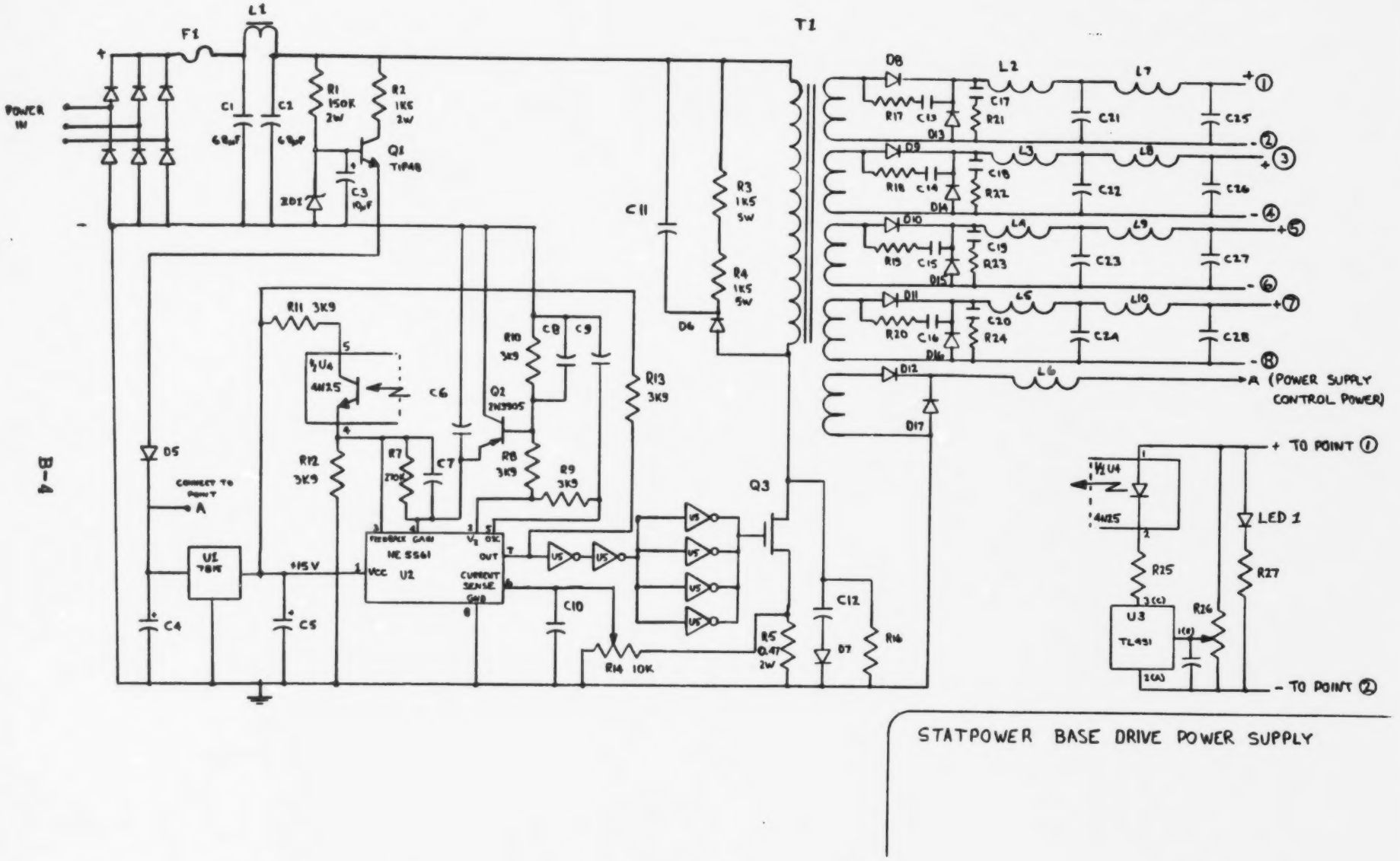
NOTES:

① Relay R1 - DPDT 1A CONTACTS
12 VDC COIL

② Adjust 10K trimmer so that ① & ② are low for input voltage less than ~80 VAC, ③ is low and ④ is high for input voltage [illegible]

Figure 19. Input Voltage Sensing Circuit, Fan Switching Circuit, and Front Panel

INPUT VOLTAGE SENSING, FAN SWITCHING AND FRONT PANEL
V H FEB 84



B-4

STATPOWER BASE DRIVE POWER SUPPLY

Figure 20. Transistor Base Drive Power Supply

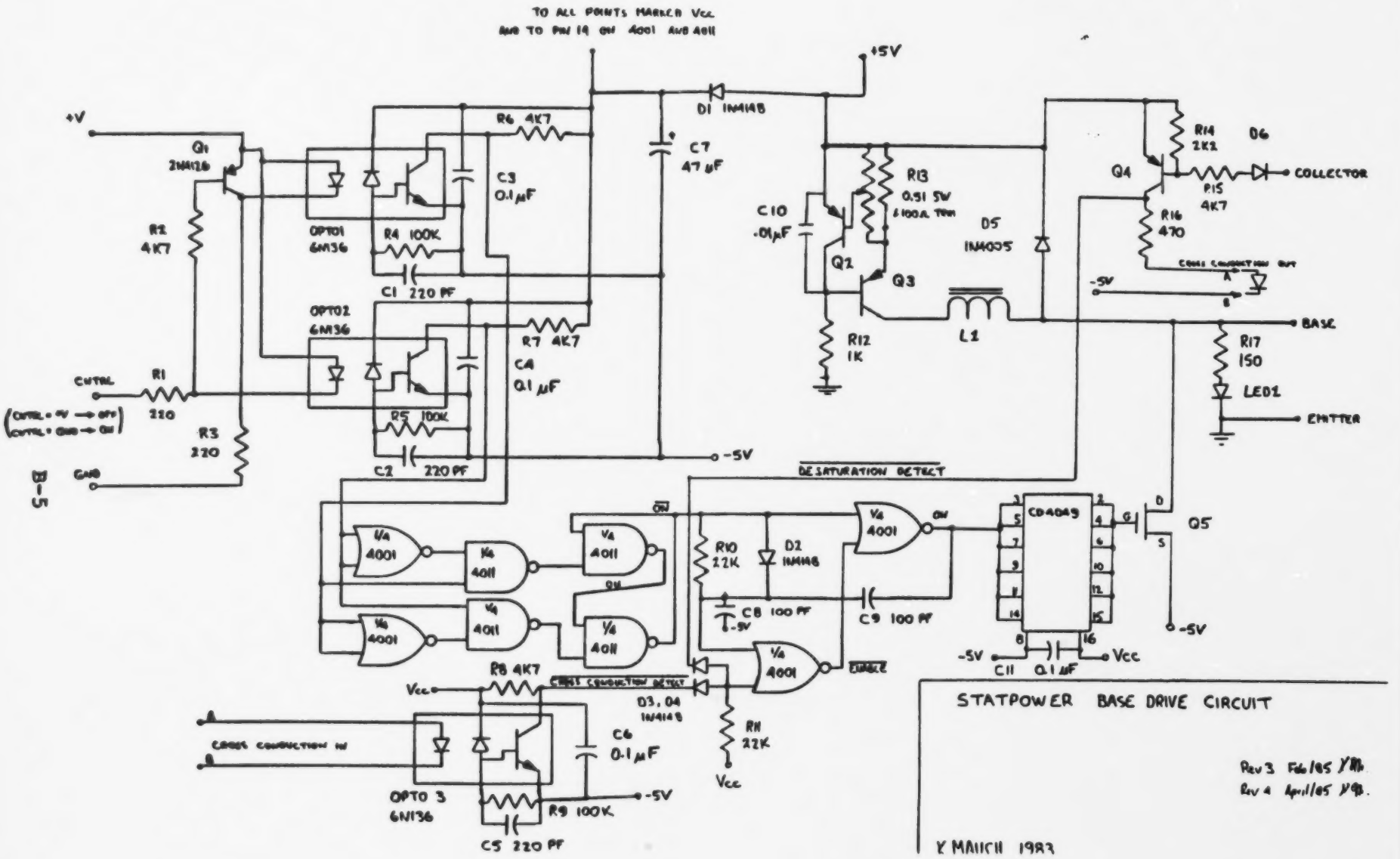


Figure 21. Transistor Base Drive Circuit

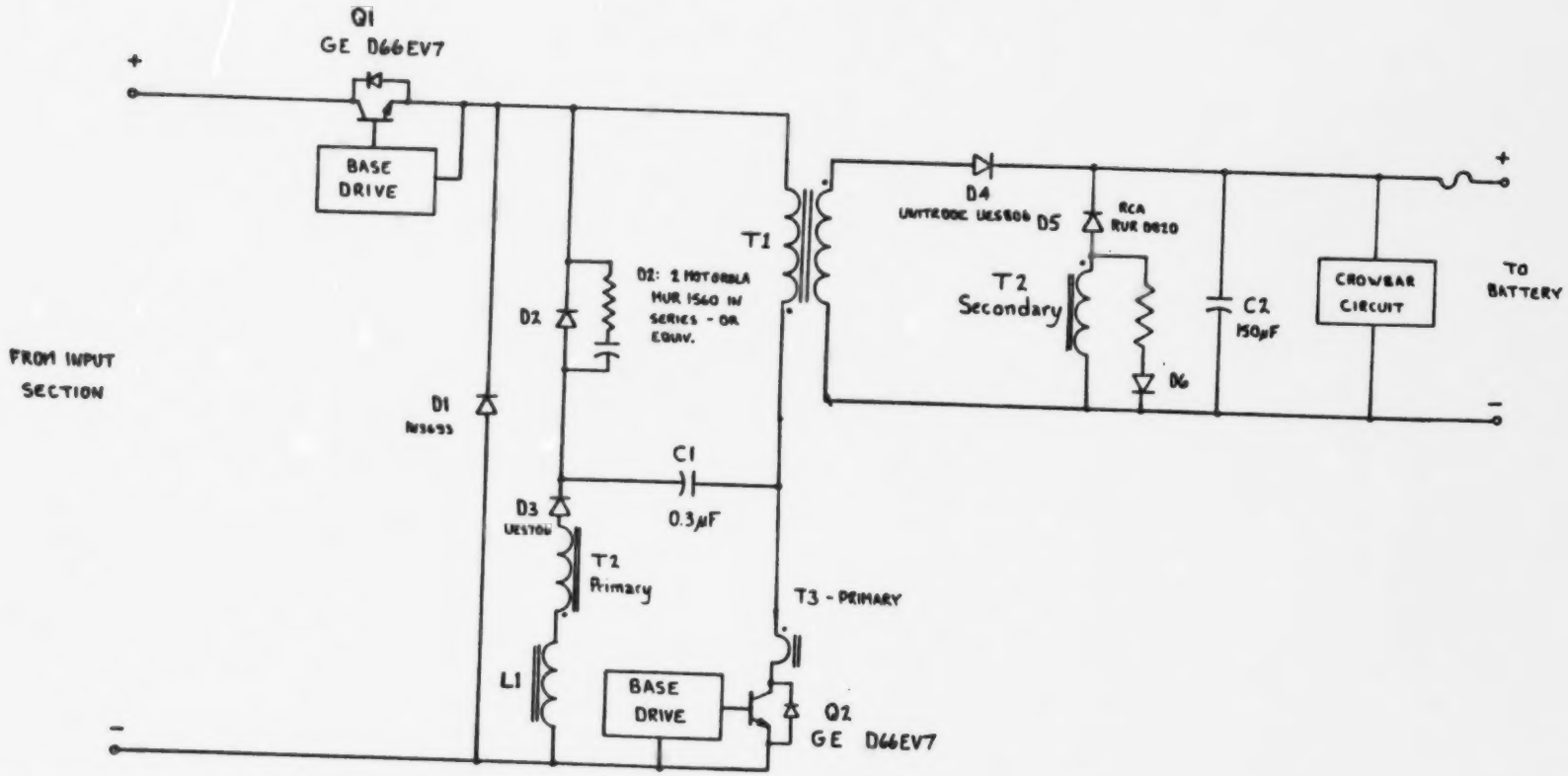
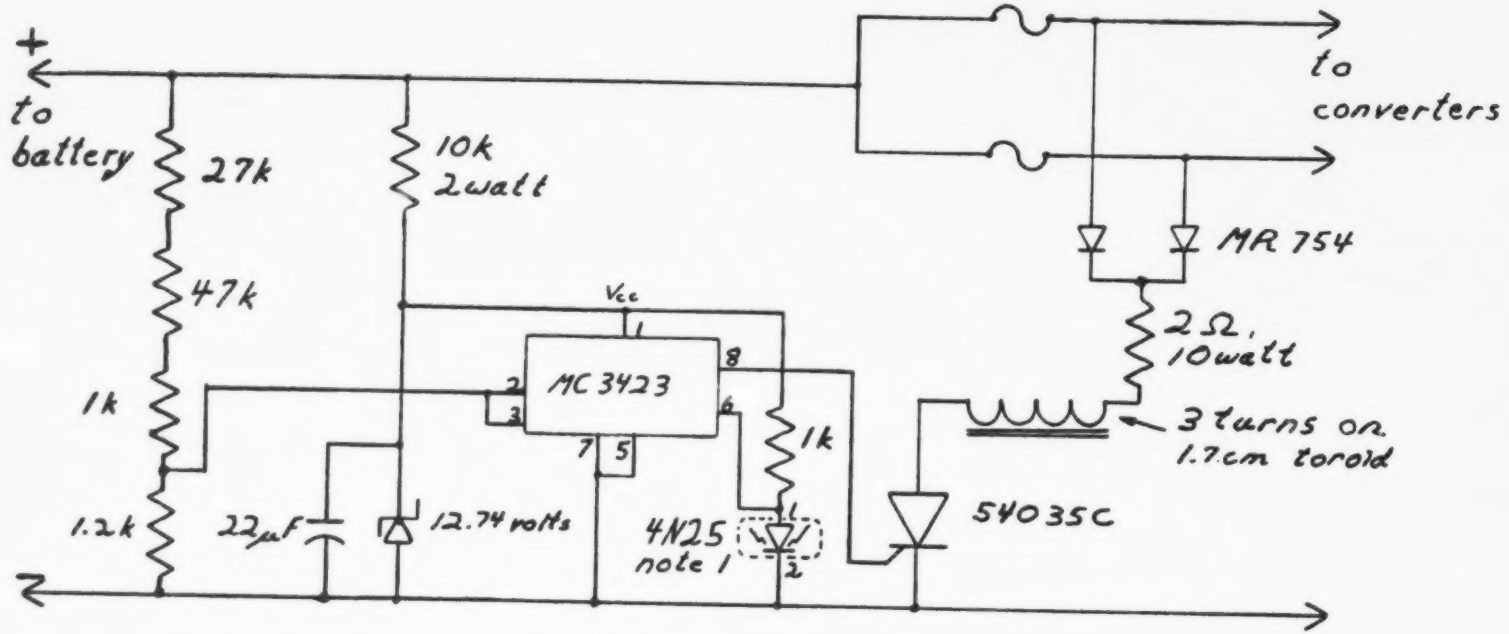


Figure 22. Charger Power Conversion Circuit



note 1 : the 4N25 circuit is expanded below

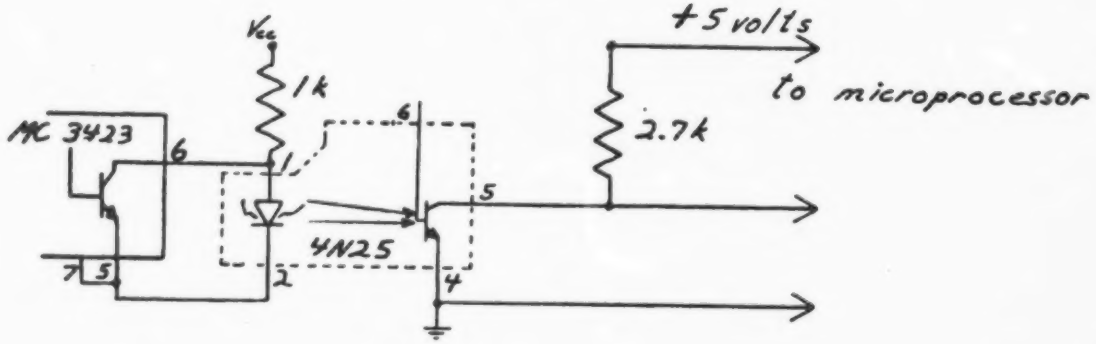


Figure 23. Output Crowbar Circuit

CROWBAR
Protection Circuit

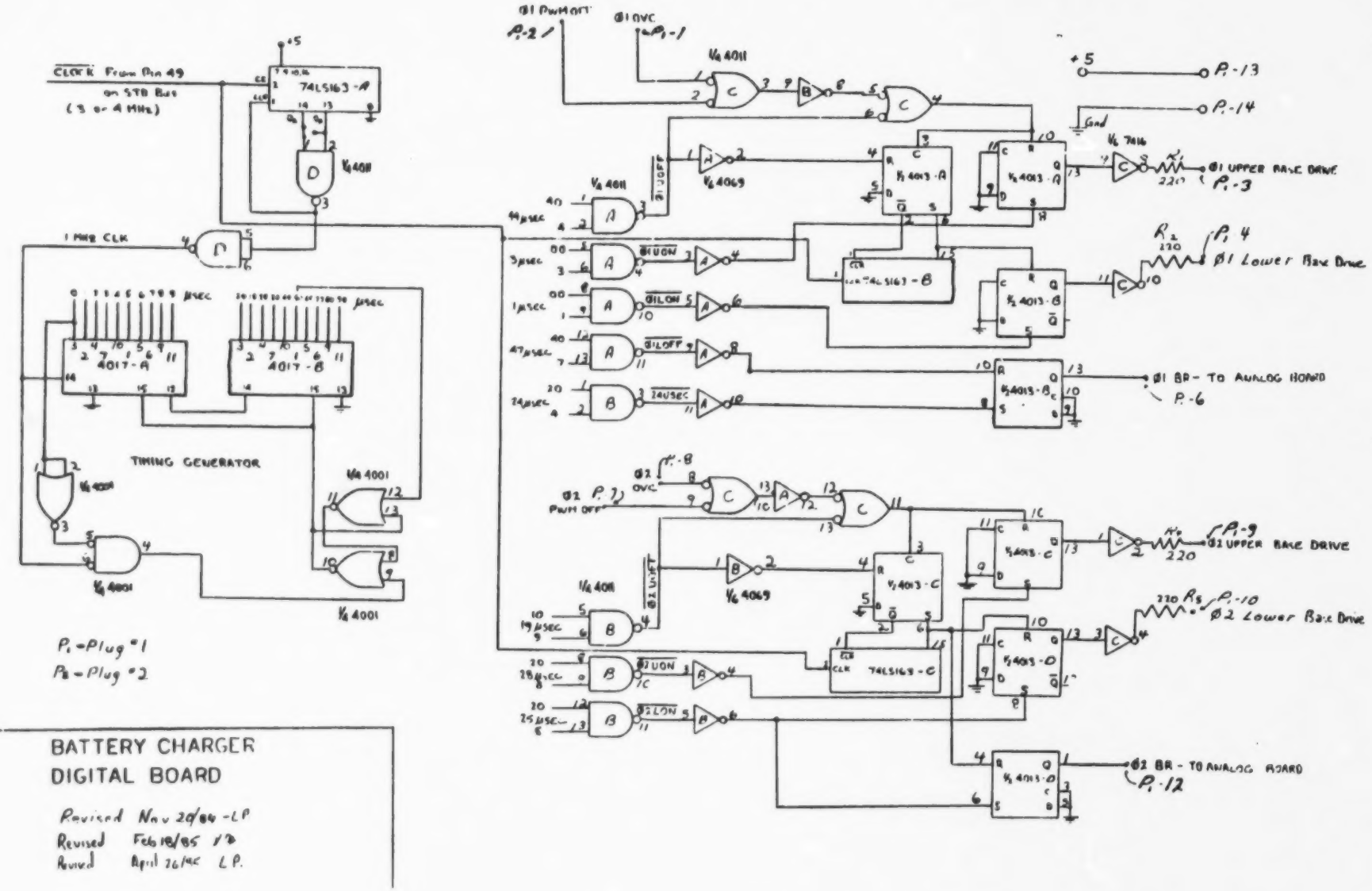


Figure 24. Inner Control Loop:
Digital Circuits

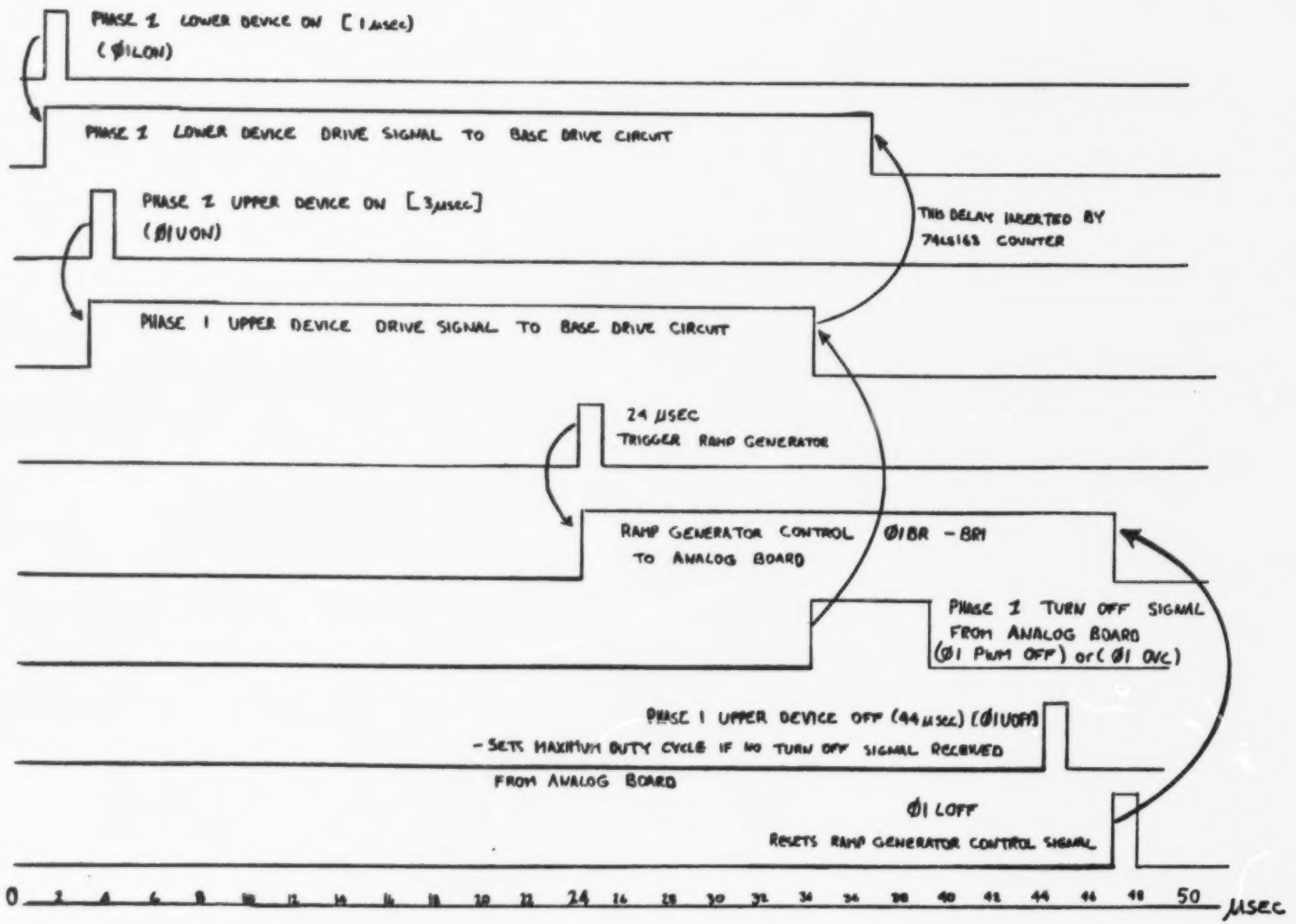

PHASE 1 LOWER DEVICE ON [1 μsec]
(ØILON)
PHASE 1 LOWER DEVICE DRIVE SIGNAL TO BASE DRIVE CIRCUIT
PHASE 1 UPPER DEVICE ON [3 μsec]
(ØIUON)
THIS DELAY INSERTED BY
74LS163 COUNTER
PHASE 1 UPPER DEVICE DRIVE SIGNAL TO BASE DRIVE CIRCUIT
24 μSEC
TRIGGER RAMP GENERATOR
RAMP GENERATOR CONTROL ØIBR - BRI
TO ANALOG BOARD
PHASE 1 TURN OFF SIGNAL
FROM ANALOG BOARD
(ØI PWM OFF) or (ØI OVC)
PHASE 1 UPPER DEVICE OFF (44 μsec) (ØIUOFF)
- SETS MAXIMUM DUTY CYCLE IF NO TURN OFF SIGNAL RECEIVED
FROM ANALOG BOARD
ØI LOFF
RESETS RAMP GENERATOR CONTROL SIGNAL
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 μsec

Figure 25. Timing Diagram For Digital Section of Inner Control Loop

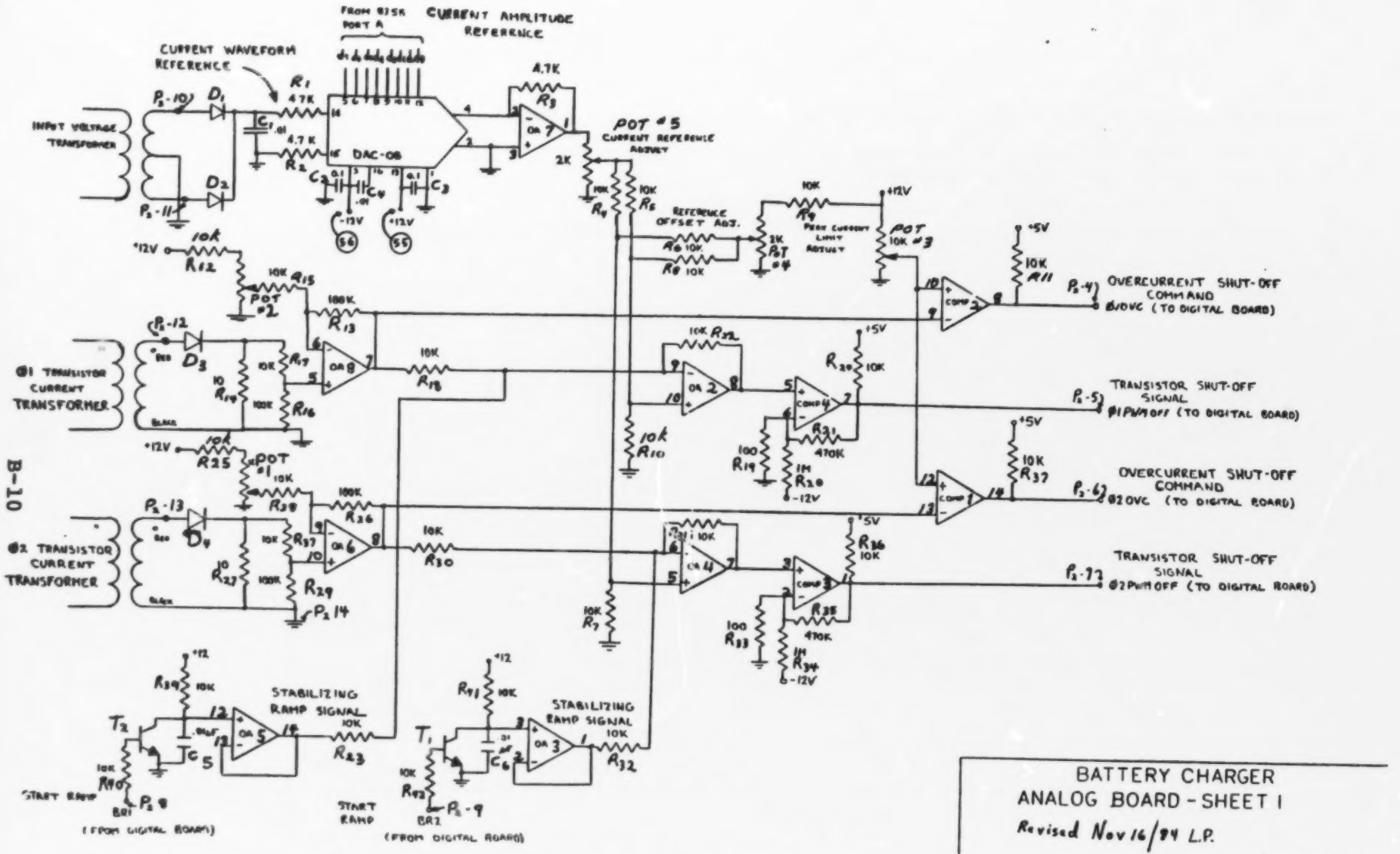


Figure 26. Inner Control Loop: Analog Circuits

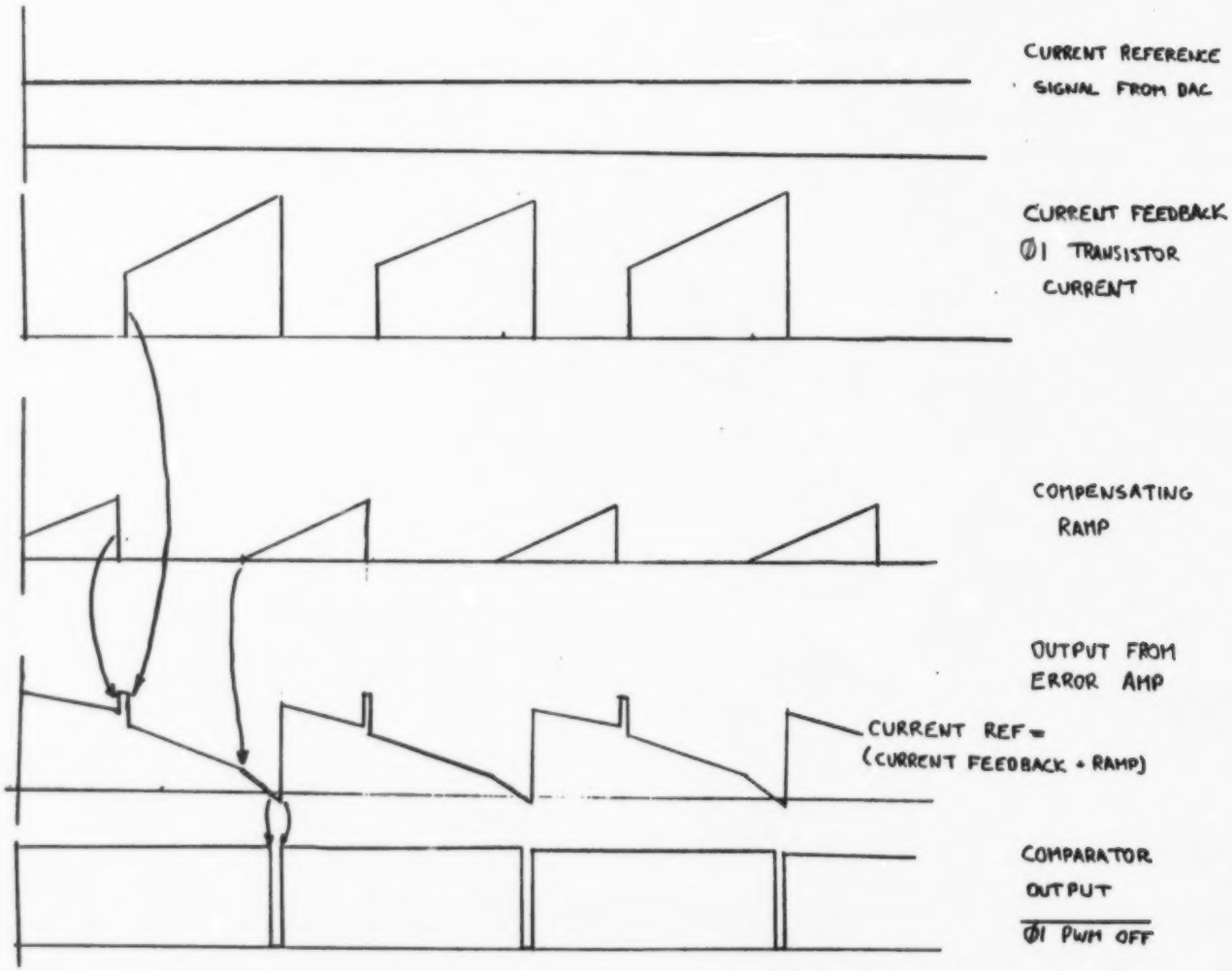


Figure 27. Waveform Diagram for Analog Section of Inner Control Loop

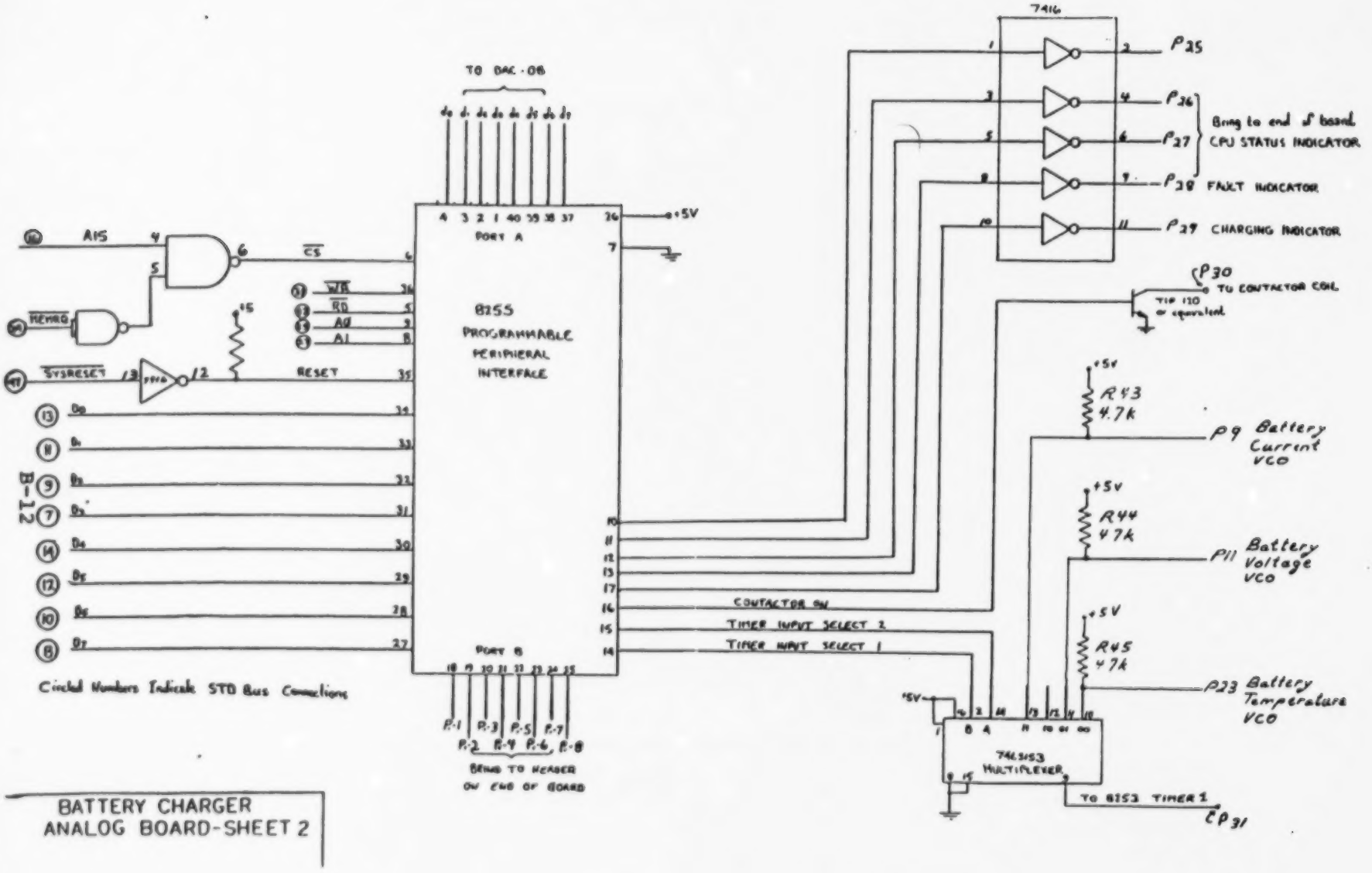


Figure 28. Interface to DAC, Sensor Signals, and Contactor

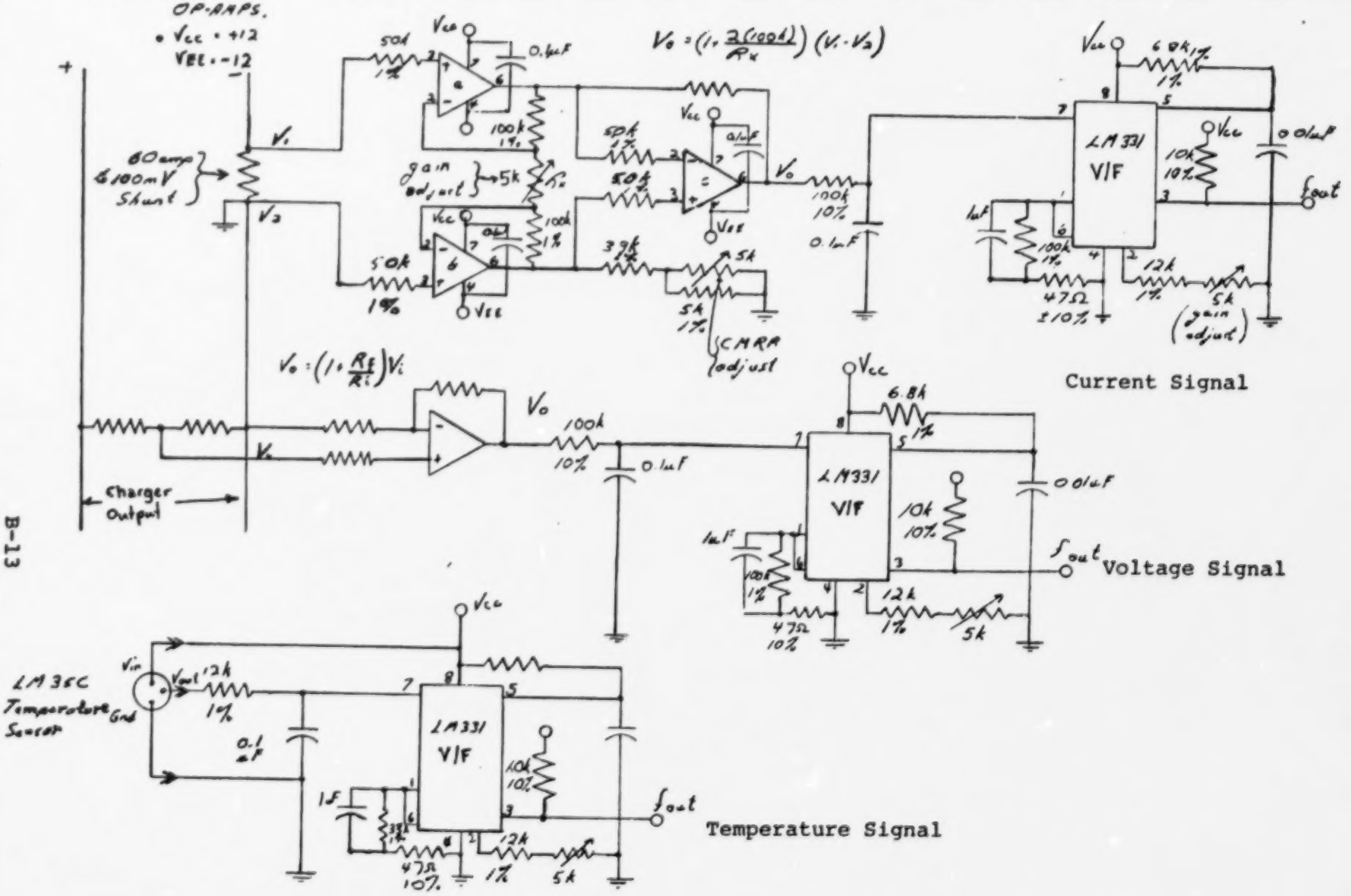


Figure 29. Feedback Signal Conditioning Circuits

APPENDIX C

PROGRAM LISTINGS

C.1 System Memory and I/O Map

Memory:

0000 - 0FFF	EPROM 0	(2732A)
1000 - 1FFF	EPROM 1	(2732A)
2000 - 2FFF	EPROM 2	(2732A)
3000 - 3FFF	EPROM 3	(2732A)
4000 - 4FFF	EPROM 4	(2732A)
7800 - 7FFF	RAM 0	(6116)

I/O:

8255 Programmable Peripheral Interface (Memory Mapped)

8000	Port A
8001	Port B
8002	Port C
8003	Control Register

8253 Programmable Counter - Timer (I/O Mapped)

7C	Timer 0
7D	Timer 1
7E	Timer 2
7F	Control Register

MDX - EPROM/UART Card (I/O Mapped)

DC	UART Data Register
DD	UART Control Register
DE	System Control Register

```
/*

                Constant  definitions */

#define         STACKTOP  0x8000        /*address of top of stack */
#define         NTASKS  3               /*number of tasks*/
#define         DORMANT 1               /*task status */
#define         ASLEEP  2               /*task status */
#define         READY   3               /* task status */
#define         RUNNING 4               /*task status*/
#define         ONESEC  50              /*timer value required for a period
                                          of one second*/
#define         NUMCHAN 2               /*number of message channels*/
#define         BUFLENGTH 64    /* length of buffer in message channel */
#define         INCHANNEL 0             /* terminal input channel */
#define         OUTCHANNEL 1            /* terminal output channel */
#define         FULL        1           /*buffer status */
#define         EMPTY       0           /* buffer status */
#define         NOTEMPTY    2           /* buffer status */

#define         FOREVER   1             /* used for continuous loops */
#define         ON          1
#define         OFF         0
#define         DAV         0x40        /* mask to check for received data */
#define         TBMT        0x80        /* mask to check for TX ready */
#define         KBD     1               /* flag to indicate display status */
#define         CPU     0               /*flag to indicate display status */
#define         SET     1               /* */
#define         FAST    12              /* constant to blink status light at 4
#define         SLOW    25              /* blink status light at 2 Hz */
#define         NOPOWER 1
#define         LOWTEMP 2
#define         PORTB   0x8001          /* 8255 port addresses  */
#define         DAC     0x8000
#define         CNTRL   0x8003
#define         UTEMP   50              /* upper temp limit for battery */
#define         HITEMP  1
#define         HIVOLTS 2
#define         LOVOLTS 3
#define         OVRTIME 4
#define         PARAMINIT 2
#define         IDLEOFF 3
#define         CONSTCURR 4
#define         CONSTVOLT 5
#define         FINISH 6
#define         EQUAL 7
#define         BLOCKED 8
#define         FAULT 9
#define         DEL1  10
#define         DEL2  11
#define         CURRENT 1
#define         VOLTS   2
#define         TEMP    3
```

```
#define         GENERIC 1
#define         EV1000  2
#define         XPV23   3
#define         EV106   4
#define         SUCCESS 1
#define         NOSUCCESS 0
#define         NL      /r/n
```

```
/*
                Initialization and Task Scheduling Module */

        #include "b:header"
        /* external variable declarations */

        int templ;      /* temporary external storage */
        int nextstate;  /* charger state */
        int faultflg;   /* indicates that charger fault has occurred */
        int initflg;    /* indicates that charger has been initialized */
        int equflg;     /* indicates that an equalization charge is being performed */
        int tmr;        /* indicates whether charge timer is active */
        int stopnow;    /* indicates that charging is to stop immediately */
        int startnow;  /* indicates that charging is to start immediately */
        int onoff;      /* charger status - on or off */
        int dtemp;      /* display temperature variable */
        int temp;       /* temperature variable */
        int dvolts;     /* display volts variable */
        int volts;      /* battery volts * 5 */
        int v1,v2,v3;   /* N-1 to N-3 battery voltage measurements */
        int dv;         /* rate of change of battery voltage */
        int dcurrent;   /* display current variable (amps) */
        int current;    /* charger output current * 10 */
        int curlimit;   /* charger output current limit */
        int gasvolts;   /* battery gassing voltage */
        int battype;    /* battery type */
        int batvolts;   /* nominal battery voltage */
        int ampsin;     /* nominal maximum input current to charger */
        int blockreason; /*reason why charging is blocked */
        int faultreason; /* reason why charger fault occurred */
        int tick;       /* keeps track of clock ticks for charge timer */
        unsigned int timer;      /* keeps track of elapsed seconds for charge timer */
        int fintime;    /* length of time to be spent during finishing charge */
        int waitcount, datacount;  /* counters */
        int datatype;   /* determines which variable is to be measured */
        int setpoint;
        int bstatus;
        char *wkdays[7]; /* array of pointers to strings containing days of week*/
        char *cntrl;    /* pointer to 8255 control register */
        char *dac;      /* pointer to DAC register */
        char *strptr;   /* pointer to comstring */
        char *command;  /*pointer to command string*/
        char comstring[20]; /*array where command string is stored */
        int(*next_item)(); /*pointer to function which executes next menu item */

        struct tcb              /* definition of task control block */
                {
                char status;
                int wait_time;
                int waitperiod;
                }
                ;
```

```
struct tcb tcb_array[NTASKS];  /* array of task control blocks*/

struct tcb *running;            /* pointer to tcb of running task */

struct chanhandler      /* definition of message channel handler structure*/
        {
        char *top;      /*pointer to top of buffer array*/
        char *head;     /*pointer to head of message channel*/
        char *tail;     /*pointer to tail of message channel */
        char *bottom;   /* pointer to bottom of buffer array */
        int chanstatus; /* status of channel */
        }
        ;

struct chanhandler *chanpoint[NUMCHAN];   /*array of pointers to channels*/

char buf1[BUFLENGTH], buf2[BUFLENGTH];          /* buffer arrays */
struct chanhandler chan1, chan2;        /* message channel handler structures*/
struct time             /* definition of structure containing time of day */
        {
        int clock_interval;
        int seconds;
        int minutes;
        int hours;
        int days;
        }
        ;

struct time global_time; /* master time updated by real-time clock interrupt */
struct time chrg_time;
struct time equ_time;

/* Function main() initializes data structures, calls initialization
function to initialize variables and hardware, then starts scheduler */
main()
{
int i;
extern int statesequencer();
extern int menu_sequencer();
int blink();

/* message channel initialization here */

chan1.top = buf1;
chan1.head = buf1;
chan1.tail = buf1;
chan1.bottom = buf1 + (BUFLENGTH - 1);
chan1.chanstatus = EMPTY;
chanpoint[0] = &chan1;
chan2.top = buf2;
chan2.head = buf2;
chan2.tail = buf2;
```

```
chan2.bottom = buf2 + (BUFLENGTH - 1);
chan2.chanstatus = EMPTY;
chanpoint[1] = &chan2;

/* tcb initialization here */

tcb_array[0].status = READY;
tcb_array[0].wait_time = 0;
tcb_array[0].waitperiod = ONESEC;

tcb_array[1].status = READY;
tcb_array[1].wait_time = 0;
tcb_array[1].waitperiod = ONESEC/5;

tcb_array[2].status = READY;
tcb_array[2].wait_time = 0;
tcb_array[2].waitperiod = SLOW;

initialize();   /*perform all required initializations before interrupts are
                  enabled */
enable();       /* enable interrupts */
sched();        /* invoke scheduler */
}

sched()  /*scheduler - schedules READY tasks in round-robin fashion */
{
int i;
while(FOREVER)  /* loops continuously */
        {
        for(i = 0; i <= NTASKS-1; i++)  /* while indexing through array of */
                {                       /* TCBs */
                if(tcb_array[i].status == READY)  /*find first READY task*/
                  {
                  tcb_array[i].wait_time = tcb_array[i].waitperiod;
                  tcb_array[i].status = RUNNING;
                  run(i);   /* activate the task */
                  }
                }
        }
}


run(i)  /* function called by sched() to transfer control from exec to task */
int i;
{
switch(i) {
        case 0:
                statesequencer();  /* Sequencer Module */
                break;
        case 1:
                menu_sequencer();  /* Control Panel Module */
                break;
```

```
        case 2:
                blink();            /* Blink CPU Status Light */
        }
disable();
if(tcb_array[i].status != READY)
        tcb_array[i].status = ASLEEP;
enable();
}


/* message handling functions */

insert_into_channel(chanmum, message)  /*adds data to head of FIFO queue*/
int chanmum;    /* number of message channel to be used */
char *message;  /* data item to be inserted into channel */
{
disable();
if(chanpoint[chanmum] -> chanstatus != FULL)  /* can only insert data */
        {                                     /* if queue isn't full */
        *(chanpoint[chanmum] -> head) =*message;  /* add data to queue */
        (chanpoint[chanmum] -> head)++; /* advance pointer to head of queue
                                           by one location */

        if((chanpoint[chanmum] -> head) > (chanpoint[chanmum] -> bottom))
          (chanpoint[chanmum] -> head) = (chanpoint[chanmum] -> top);

/* above ensures that queue does not grow beyond limits of buffer  */

/* check whether the buffer is full and set status accordingly */

        if((chanpoint[chanmum] -> head) == (chanpoint[chanmum] -> tail))
          (chanpoint[chanmum] -> chanstatus) = FULL;
        else
          (chanpoint[chanmum] -> chanstatus) = NOTEMPTY;

        return(NOTEMPTY);
        }
else

        return(FULL);
}

get_from_channel(chanmum, message)   /* get data from tail of FIFO queue */
int chanmum;   /* number of message channel to be used */
char *message; /* pointer to location where retrieved data is to be stored */
{
disable();
if(chanpoint[chanmum] -> chanstatus != EMPTY)  /*check to see if queue is empty */
        {                                      /* if not ... */
        *message = *(chanpoint[chanmum] -> tail);  /* get data item from
                                                      queue tail */
        (chanpoint[chanmum] -> tail)++;         /* adjust pointer to tail */
```

```
        if((chanpoint[chanmum] -> tail) > (chanpoint[chanmum] -> bottom))
          (chanpoint[chanmum] -> tail) = (chanpoint[chanmum] -> top);

/* above ensures that tail of queue "wraps around" when bottom of circular
   buffer is reached  */

/* check to see if queue is now empty and set status accordingly */
        if((chanpoint[chanmum] -> tail) == (chanpoint[chanmum] -> head))
          (chanpoint[chanmum] -> chanstatus) = EMPTY;
        else
          (chanpoint[chanmum] -> chanstatus) = NOTEMPTY;
        return(NOTEMPTY);
        }
else
        return(EMPTY);  /* return this if queue was originally empty */
}

/* Initialization function to initialize internal variables and
hardware registers to defined state when system is powered up
or reset.     */

initialize()
{
int menu1();
faultflg = initflg = equflg = tmr = stopnow = startnow = OFF;
onoff = ON;
nextstate = PARAMINIT;
strptr = comstring;
next_item = menu1;
wkdays[0] = "SUNDAY";
wkdays[1] = "MONDAY";
wkdays[2] = "TUESDAY";
wkdays[3] = "WEDNESDAY";
wkdays[4] = "THURSDAY";
wkdays[5] = "FRIDAY";
wkdays[6] = "SATURDAY";
dtemp = temp = dvolts = volts = v1 = v2 = v3 = dv = dcurrent = current =
curlimit = gasvolts = fintime = waitcount = datacount = timer = tick =
setpoint = bstatus = 0;
datatype = VOLTS;
cntrl = CNTRL;
*cntrl = 0x82; /* set up 8255 with port A output, port B input, port C output
                  - all in Mode 0 */
dac = DAC;
*dac = 0;
*cntrl = 0;     /* select voltage sensor */
*cntrl = 3;
*cntrl = 4;     /* open contactor */
*cntrl = 6;     /* turn off indicators */
*cntrl = 8;
global_time.clock_interval = chrg_time.clock_interval = 0;
```

```
equ_time.clock_interval = global_time.seconds = chrg_time.seconds = 0;
equ_time.seconds = global_time.minutes = chrg_time.minutes = 0;
equ_time.minutes = global_time.hours = chrg_time.hours = 0;
equ_time.hours = global_time.days = chrg_time.days = equ_time.days = 0;
init_uart();
init_timer();
sim(0x19);      /* Unmask RST7.5 and RST6.5 */
}

init_uart()   /* initialize UART to 1 stop bit, even parity, 7 data bits */
{
#asm
        MVI A, 30H      ; data format to UART control port
        OUT 0DDH
        MVI A, 3        ; CTS, DSR lines active to system control port
        OUT 0DEH
#endasm
}

init_timer()    /* set up 8253 timer */
{
#asm
        MVI A, 34H      ;TIMER 0 IN MODE 2 (RATE GENERATOR)
        OUT 7FH
        MVI A, 70H      ; TIMER 1 IN MODE 0
        OUT 7FH
        MVI A, 0B0H     ; TIMER 2 IN MODE 0
        OUT 7FH
        MVI A, 88H      ; LSB OF TIMER 0 CONTENTS (5000 DECIMAL)
        OUT 7CH
        MVI A, 13H      ; MSB OF TIMER 0 CONTENTS
        OUT 7CH
        MVI A, 0A8H     ; LSB OF TIMER 1 CONTENTS (25000 DECIMAL)
        OUT 7DH
        MVI A, 61H      ; MSB OF TIMER 1 CONTENTS
        OUT 7DH
        MVI A, 0E8H     ; LSB OF TIMER 2 CONTENTS (1000 DECIMAL )
        OUT 7EH
        MVI A, 3        ; MSB OF TIMER 2 CONTENTS
        OUT 7EH
#endasm
}


/* Blinks CPU Status Light by toggling every time it is called */

blink()
{
if(bstatus == ON)
        {
        bstatus = OFF;
        *cntrl = 0xA;   /* Turn lamp off (direct Port C control) */
```

```
            }
        else
            {
            bstatus = ON;
            *cntrl = 0xB;  /* Turn lamp on  (direct Port C control */
            }
        }
        /*
```

```
/*

                State sequencer for battery charger */

#include "b:header"
#define TRUE 1
#define FALSE 0
#define OVRVOLTS    ((batvolts * 15)/2))   /* upper voltage limit
                                                - 3 volts/cell   */

/* The variable nexstate defines the next state the charger is
   to enter.   The statesequencer function invokes the function
   which carries out the actions required in that state.  */

statesequencer()
{
        if(faultflg == SET)
                nextstate = FAULT;
        switch(nextstate){
                case PARAMINIT:
                        paraminit();
                        break;
                case IDLEOFF:
                        idle_off();
                        break;
                case DEL1:
                        del1();
                        break;
                case CONSTCURR:
                        constcurr();
                        break;
                case DEL2:
                        del2();
                        break;
                case CONSTVOLT:
                        constvolt();
                        break;
                case FINISH:
                        finish();
                        break;
                case EQUAL:
                        equal();
                        break;
                case BLOCKED:
                        blocked();
                        break;
                case FAULT:
                        fault();
                        break;
                default:
                        fault;
                }
}
```

```
paraminit()                    /* Parameter Initialization */
{
if (initflg == SET)
        {
        tcb_array[2].waitperiod = FAST;
        nextstate = IDLEOFF;
        }
}

idle_off()                     /* Idle/Off */
{
stopnow = OFF;
disable();
dtemp = temp;
dvolts = volts/5;              /* get updated voltage, current, and   */
dcurrent = 0;                  /* temperature */
tmr = OFF;
enable();

/* Code below checks to see if charging should start and if
   power is connected and battery temp > zero  */

if( (onoff == ON) && ( starttime() || ( startnow == ON)))
        {
        if( power() == 0)
                {
                blockreason = NOPOWER;
                nextstate = BLOCKED;
                }
        else if( dtemp <2 )
                {
                blockreason = LOWTEMP;
                nextstate = BLOCKED;
                }
        else
                {
                startcharge();          /* get ready to start charging  */
                nextstate = DEL1;
                }
        }
}

/* Checks to see if present time equals charging start time or
   equalization charge start time  */

starttime()
{
struct time ptime;

gettime(&ptime);
if((ptime.days != equ_time.days) && (ptime.hours == chrg_time.hours)
```

```
 && (ptime.minutes == chrg_time.minutes))
                {
                equflg = OFF;
                return(TRUE);
                }
if((ptime.days == equ_time.days) && (ptime.hours == equ_time.hours)
 && (ptime.minutes == equ_time.minutes))
        {
        equflg = ON;
        return(TRUE);
        }
return(FALSE);
}

/* Checks for presence of AC power and determines whether its
   120 or 240 volts   */

power()
{
char *portb;
int pwr;
portb = PORTB;
switch(*portb & 3){     /*mask off two lower order bits in port B */
        case 0:
                pwr = 0;
                break;
        case 2:
                pwr = 120;
                break;
        case 3:
                pwr = 240;
                break;
        default:
                pwr = 0;
                break;
        }
return(pwr);
}

/* Performs housekeeping required before charging can start */

startcharge()
{
disable();
tick = 0;       /* start timer which times length of charge */
timer = 0;
tmr = ON;
enable();
current = 0;    /* initialize value of current prior to first measurement */
setpoint = 0;
*dac = 0;       /* initialize DAC output */
curlimit = clim();  /*calculate current setpoint */
```

```
*cntrl = 7;      /* Turn on Charging Lamp */
*cntrl = 5;      /* Turn on Contactor */
}

/* Determine maximum charger output current  - depends on power capacity
   of AC supply    */

clim()
{
int limit;
if(power() == 120)
        limit = (ampsin * 30)/4;
else
        limit = (ampsin * 15);
return(limit);
}

/* One second delay */

del1()
{
nextstate = CONSTCURR;
}

/* Constant Current Charge  */

constcurr()
{
unsigned int loctimer;
int locvolts;
disable();
dcurrent = current/10;
dvolts = volts/5;
locvolts = volts;
dtemp = temp;
loctimer = timer;
enable();
startnow = OFF;
if ((stopnow == TRUE)||(onoff == OFF))  /* Stop charging ? */
        {
        if(loctimer <61)
          nextstate = DEL2;
        else
          nextstate = IDLEOFF;
        shutdown();
        }
else if( power() == 0)                  /* AC power present ? */
        {
        blockreason = NOPOWER;
        nextstate = BLOCKED;
        shutdown();
        }
```

```
else if( dtemp < 2)
        {
        blockreason = LOWTEMP;
        nextstate = BLOCKED;
        shutdown();                         /* Battery temp. OK ? */
        }
else if(dtemp > UTEMP)
        {
        faultreason = HITEMP;
        nextstate = FAULT;
        shutdown();
        }
else if(locvolts > OVRVOLTS)
        {
        faultreason = HIVOLTS;
        nextstate = FAULT;
        shutdown();                         /* Battery volts within */
        }                                   /*   limits ?      */
else if(locvolts < (batvolts * 4))
        {
        faultreason = LOVOLTS;
        nextstate = FAULT;
        shutdown();
        }
else if((loctimer/60)*(curlimit/10) > 12000)
        {
        faultreason = OVRTIME;                      /* charging time   */
        nextstate = FAULT;                  /* less than max allowable ? */
        shutdown();
        }
else if(locvolts > (gasvolts = gvolts(dtemp)))  /* battery voltage    */
        {                                   /*   greater than gassing     */
        nextstate = CONSTVOLT;              /*    voltage ?               */
        }
return;
}

/* Calculate battery gassing voltage */

gvolts(temperature)
int temperature;
{
int gv, n;
n = batvolts/6;       /* number of 6 volt batteries in series */
switch(battype){
        case EV1000:
                gv = (n*43) - ((n*37*temperature)/4);
                break;
        case EV106:
        case XPV23:
        case GENERIC:
                gv = (n*37) - ((n*temperature)/11);
```

```
                break;
        }
return(gv);
}

/*  Delay used to ensure that charger doesn't immediately start again
    if charging is stopped less than a minute after programmed start
    time   */

del2()
{
unsigned int loctimer;
disable();
loctimer = timer;
enable();
if(loctimer > 60)
        nextstate = IDLEOFF;
}

/*  Constant Voltage Charge   */

constvolts()
{
unsigned int loctimer;
int locvolts;
disable();
loctimer = timer;
dcurrent = current/10;
dtemp = temp;
locvolts = volts;
gasvolts = gvolts(dtemp);
enable();
dvolts = locvolts/5;
if ((stopnow == TRUE)||(onoff == OFF))
        {
        if(loctimer <61)
          nextstate = DEL2;
        else
          nextstate = IDLEOFF;
        shutdown();
        }
else if( power() == 0)
        {
        blockreason = NOPOWER;
        nextstate = BLOCKED;
        shutdown();
        }
else if( dtemp < 2)
        {
        blockreason = LOWTEMP;
        nextstate = BLOCKED;
        shutdown();
```

```
        }
else if(dtemp > UTEMP)
        {
        faultreason = HITEMP;
        nextstate = FAULT;
        shutdown();
        }
else if(locvolts > OVRVOLTS)
        {
        faultreason = HIVOLTS;
        nextstate = FAULT;
        shutdown();
        }
else if(locvolts < (batvolts * 4))
        {
        faultreason = LOVOLTS;
        nextstate = FAULT;
        shutdown();
        }
else if(loctimer > 36000)
        {
        faultreason = OVRTIME;
        nextstate = FAULT;
        shutdown();
        }
else if(dcurrent <= 5)                  /* Has charging current dropped */
        {                               /* below 5 amperes ?            */
        if(equflg == ON)
                nextstate = EQUAL;
        else
                {
                nextstate = FINISH;     /* Calculate length of finishing  */
                if(loctimer < 3600)         /* charge  */
                    fintime = loctimer/10;
                else
                    fintime = 10800;
                }
        disable();
        timer = 0;
        enable();
        }

}

/*  Finishing Charge   */

finish()
{
unsigned int loctimer;
int locvolts;
disable();
loctimer = timer;
```

```
dcurrent = current/10;
dtemp = temp;
locvolts = volts;
enable();
dvolts = locvolts/5;
if ((stopnow == TRUE)||(onoff == OFF))
        {
        if(loctimer <61)
          nextstate = DEL2;
        else
          nextstate = IDLEOFF;
        shutdown();
        }
else if( power() == 0)
        {
        blockreason = NOPOWER;
        nextstate = BLOCKED;
        shutdown();
        }
else if( dtemp < 2)
        {
        blockreason = LOWTEMP;
        nextstate = BLOCKED;
        shutdown();
        }
else if(dtemp > UTEMP)
        {
        faultreason = HITEMP;
        nextstate = FAULT;
        shutdown();
        }
else if(locvolts > OVRVOLTS)
        {
        faultreason = HIVOLTS;
        nextstate = FAULT;
        shutdown();
        }
else if(locvolts < (batvolts * 4))
        {
        faultreason = LOVOLTS;
        nextstate = FAULT;
        shutdown();
        }
else if(loctimer > fintime)             /* End of charge time ? */
        {
        nextstate = IDLEOFF;
        shutdown();
        }
}

/*  Equalizing Charge   */
```

```
equal()
{
unsigned int loctimer;
int locvolts;
disable();
loctimer = timer;
dcurrent = current/10;
dtemp = temp;
locvolts = volts;
enable();
dvolts = locvolts/5;
if ((stopnow == TRUE)||(onoff == OFF))
        {
        if(loctimer <61)
          nextstate = DEL2;
        else
          nextstate = IDLEOFF;
        shutdown();
        }
else if( power() == 0)
        {
        blockreason = NOPOWER;
        nextstate = BLOCKED;
        shutdown();
        }
else if( dtemp < 2)
        {
        blockreason = LOWTEMP;
        nextstate = BLOCKED;
        shutdown();
        }
else if(dtemp > UTEMP)
        {
        faultreason = HITEMP;
        nextstate = FAULT;
        shutdown();
        }
else if(locvolts > OVRVOLTS)
        {
        faultreason = HIVOLTS;
        nextstate = FAULT;
        shutdown();
        }
else if(locvolts < (batvolts * 4))
        {
        faultreason = LOVOLTS;
        nextstate = FAULT;
        shutdown();
        }
else if(loctimer > 18000)       /* equalization = 5 hrs */
        {
        nextstate = IDLEOFF;
```

```
        shutdown();
        }
}

/*  Blocked State */

blocked()
{
unsigned int loctimer;
int locvolts;
disable();
loctimer = timer;
dcurrent = current/10;
dtemp = temp;
locvolts = volts;
tmr = OFF;
enable();
dvolts = locvolts/5;
if ((stopnow == TRUE)||(onoff == OFF))
        {
        if(loctimer <61)
          nextstate = DEL2;
        else
          nextstate = IDLEOFF;
        shutdown();
        }
else if((power() != 0) && (dtemp >= 2 && dtemp < UTEMP))
        {
        disable();
        tmr = ON;               /* if power comes on or battery temp.  */
        enable();               /* rises above 0, start charging       */
        *dac = 0;
        current = 0;
        *cntrl = 5;
        *cntrl = 7;
        nextstate = DEL1;
        }
else if(dtemp > UTEMP)
        {
        faultreason = HITEMP;
        nextstate = FAULT;
        shutdown();
        }
else if(locvolts > OVRVOLTS)
        {
        faultreason = HIVOLTS;
        nextstate = FAULT;
        shutdown();
        }
else if(locvolts < (batvolts * 4))
        {
        faultreason = LOVOLTS;
```

```
        nextstate = FAULT;
        shutdown();
        }
}

/*  Fault  State  */

fault()
{
*cntrl = 9;           /*  Light Fault indicator lamp  */
}

/*  Actions required to stop charging  */

shutdown()
{
setpoint = 0;
*dac = 0;       /* set DAC to zero */
*cntrl = 4;     /* open Contactor  */
*cntrl = 6;     /* turn off Charging indicator */
}
```

```
*/

                /*   Real - Time Clock Module   */

/* Interrupt service routine (RST7.5) for real-time clock interrupt.
     Interrupt occurs every 20 milliseconds (50 Hz)    */

clocktick()
{
int i;
/* routine first decrements the time variable of all TCBs whose status
   is ASLEEP or RUNNING.  If a time variable reaches 0, the TCB status
   is changed to READY   */
for(i = 0; i < NTASKS; i++)    /*index through array of TCBs */
  {
  if((tcb_array[i].status != DORMANT) && (tcb_array[i].status != READY))
        {
        if(--(tcb_array[i].wait_time) <= 0)
            tcb_array[i].status = READY;
        }
  }
if(tmr == ON)  /* if charge timer is enabled */
        {
        tick++;                    /* update timer */
        if(tick >= ONESEC)
          {
           tick = 0;
           timer++;
          }
        }
update_time();     /* update internal clock */
check_term();   /* monitor RS-232 interface */
return;
}

/* Functions dealing with time of day */

/* returns time of day to calling function */

gettime(timeptr)
struct time *timeptr;   /*pointer to local time structure */
{
disable();  /* disable interrupts so that time data isn't corrupted */
timeptr -> seconds = global_time.seconds;
timeptr -> minutes = global_time.minutes;
timeptr -> hours = global_time.hours;
timeptr -> days = global_time.days;
enable();
}

/* Sets time of day when passed data by calling function */
```

```
settime(day, hrs, mins, secs)
int day, hrs, mins, secs;
{
disable();
global_time.days = day;
global_time.hours = hrs;
global_time.minutes = mins;
global_time.seconds = secs;
enable();
}

/* sets present day only */

setday(day)
int day;
{
disable();
global_time.days = day;
enable();
}

/* sets present hour only */

sethour(hour)
int hour;
{
disable();
global_time.hours = hour;
enable();
}

setminute(min)
int min;
{
disable();
global_time.minutes = min;
enable();
}

/* Called by the RST7.5 interrupt service routine (clocktick(). Updates
   the data structure containing the present time  */

update_time()
{
++global_time.clock_interval;
if(global_time.clock_interval == ONESEC)
        {
        global_time.clock_interval = 0;
        ++global_time.seconds;
        }
else
        return;
```

```
if(global_time.seconds == 60)
        {
        global_time.seconds = 0;
        ++global_time.minutes;
        }
else
        return;
if(global_time.minutes == 60)
        {
        global_time.minutes = 0;
        ++global_time.hours;
        }
else
        return;
if(global_time.hours == 24)
        {
        global_time.hours = 0;
        ++global_time.days;
        }
else
        return;
if(global_time.days == 7)
        {
        global_time.days = 0;
        }
return;
}

/* Terminal handler routine called by RST7.5 interrupt service
   routine (clocktick()).  Handles transfer of data between portable
   terminal (control panel) and battery charger.  */

check_term()
{
char c;
if(termstatus() & DAV)          /* Is there data from the terminal ? */
        {
        if((c = rx()) != '\r')   /* Is it a carriage return ? */
          {
          if(c == '\n')          /* Is it a line feed ? */
                return;
          else
                insert_into_channel(INCHANNEL,&c);
          }
         else
          {
          c = '\0';                                 /* C end of string marker */
          insert_into_channel(INCHANNEL,&c);
          }
         }
if(termstatus() & TBMT)      /* Is the UART able to transmit data to the
                                terminal ? */
```

```
                {
                if(get_from_channel(OUTCHANNEL,&c) != EMPTY)
                        tx(c);    /* send data to UART */
                }
}

termstatus()            /* Return contents of UART status register */
{
#asm
        IN ODDH
        MOV L,A
        MVI H, 0
#endasm
}

rx()                    /* Return contents of UART Rx data register */
{
#asm
        IN ODCH
        MOV L,A
        MVI H,0
#endasm
}

tx(c)                   /* Send data to UART for transmission */
char c;
{
#asm
        POP H
        POP D
        PUSH D
        PUSH H
        MOV A,E
        OUT ODCH
#endasm
}

flush(channum)          /* flush circular buffer */
 int channum;
{
chanpoint[channum] -> head = chanpoint[channum] -> top;
chanpoint[channum] -> tail = chanpoint[channum] -> top;
chanpoint[channum] -> chanstatus = EMPTY;
}
```

```
/*
                              Control Loop Module

        Interrupt 6.5 service routine which controls measurement of
         battery parameters and controls charging current */

        #include "b:header"

        measure()
        {
         unsigned int tim1, tim2;

        tim1 = rd_tim1();                /* Read Counter #1 */
        tim2 = rd_tim2();                /* Read Counter #2 */
        if(tim1 == 0 || (tim1 > 0xFFE6))  /* check for delayed interrupt response */
           {
           if(datatype == CURRENT)
                current = 1000 - tim2;
           else if (datatype == VOLTS)
                {
                v3 = v2;                          /* convert measured data to */
                v2 = v1;                          /* internal format   */
                v1 = volts;
                volts = 1000 - tim2;
                dv = (volts - v3 + 3*v1 - 3*v2);
                }
           else
                temp = (1000 - tim2)/25;

        /* Following code adjusts the current amplitude command (setpoint) to
           follow charging profile as defined by curlimit (current limit) and
           gassing voltage  */

           switch(nextstate) {
                case CONSTCURR:
                case FINISH:
                case EQUAL:
                        if(datatype == CURRENT)
                        {
                        if((current > curlimit) && (setpoint > 0))
                          *dac = --setpoint;
                        else if((current < (curlimit-10)) && (setpoint < 255))
                           *dac = ++setpoint;
                        }
                        datacount++;
                        if(datacount == 100)
                           {
                           datacount = 0;          /* schedules the measurements so */
                           datatype = TEMP;        /* that current is sampled 8   */
                           }                       /* a second, voltage twice a    */
                        else if(datacount%5 == 0)  /* second, and temperature once */
                           datatype = VOLTS;       /* ten seconds    */
```

```
                else
                   datatype = CURRENT;
                break;
        case CONSTVOLT:
                if (datatype == VOLTS)
                   {
                   if(((volts > gasvolts)&&(dv>=0)) || (current > curlimit))
                        {
                        if (setpoint > 0)
                                *dac = --setpoint;
                        }
                   else if((volts < (gasvolts - 5)) && (dv <= 0))
                        {
                        if (setpoint < 255)
                                *dac = ++setpoint;
                        }
                   }
                datacount++;
                if(datacount == 100)
                  {
                  datacount = 0;
                  datatype = TEMP;                  /* Schedules measurements  */
                  }                                 /* so that voltage and     */
                else if(datacount % 2 == 0)         /* current are measured 5 *
                     datatype = VOLTS;              /* times a second and      */
                else                                /* temperature once every */
                     datatype = CURRENT;            /* 10 seconds  */
                break;
        default:
                datacount++;
                if(datacount == 100)
                  {
                  datacount = 0;
                  datatype = TEMP;
                  }
                else
                  datatype = VOLTS;
                break;
        }
        }
if(datatype == VOLTS)
        {
        *cntrl = 0;                 /* if voltage measurement scheduled  */
        *cntrl = 3;                 /* set up mux to select voltage signal */
        }
else if (datatype == CURRENT)
        {                           /* if currnt measurement scheduled ... */
        *cntrl = 1;
        *cntrl = 3;
        }
else
        {                           /* temperature measurement ...   */
```

```
        *cntrl = 0;
        *cntrl = 2;
        }
ld_tim1();                          /* start Counter #1 and Counter #2  */
ld_tim2();
}

rd_tim1()                           /* Read contents of 8253 Counter #1 */
{
#asm
        MVI A, 40H
        OUT 7FH
        IN 7DH
        MOV L,A
        IN 7DH
        MOV H,A
#endasm
}

rd_tim2()                           /* Read contents of 8253 Counter #2 */
{
#asm
        MVI A, 80H
        OUT 7FH
        IN 7EH
        MOV L,A
        IN 7EH
        MOV H,A
#endasm
}

ld_tim1()                           /* start Counter # 1  */
{
#asm
        MVI A, 70H      ;PUT TIMER BACK INTO MODE 0
        OUT 7FH
        MVI A, 0A8H
        OUT 7DH
        MVI A, 61H
        OUT 7DH
#endasm
}

ld_tim2()                           /* start Counter # 2 */
{
#asm
        MVI A, 0B0H     ; PUT TIMER BACK IN MODE 0
        OUT 7FH
        MVI A, 0E8H
        OUT 7EH
        MVI A, 3H
        OUT 7EH
```

```
#endasm
}
```

```
/*
                        Control Panel Module        */

/* constant definitions here */
#include "b:header"
#define   NOCOMMAND  'Z'

/* menu_sequencer gets commands from input circular buffer and passes
   them to function carrying out actions for current menu item.  Variable
   next_item is a pointer to the function which carries out the actions
   for the current menu item     */

menu_sequencer()
{
char *get_command();
command = get_command();
(*next_item)(command);
}

/* get_command gets characters from the input circular buffer until
   it gets a '\0', signifying the end of a command, or until the
   buffer is empty.  If it gets a complete command it returns a pointer
   to the command string, otherwise it returns a pointer to the string
   "Z" signifying that no command is available */

char *get_command()
{
while(get_from_channel(INCHANNEL,strptr) != EMPTY)  /* if buffer not empty */
        {
        enable();
        if(*strptr == '\0')              /* check for end of command  string */
                {
                strptr = comstring;      /* reset strptr */
                return(strptr);          /* return pointer to command string */
                }
        else
                {
                if(*strptr == 'Z')
                        {
                        strptr = comstring;
                        return("X");
                        }
                if(++strptr == (comstring+19))          /* get ready for
                                                           next character */
                        strptr = comstring;
                }
        }
enable();
return("Z");                    /*return string indicating no command */
}

/* Function disp() is passed a pointer to a string of characters which
```

```
   are to be displayed on the control panel.  It transfers the string
   to the output circular buffer  */

disp(string)
char *string;
{
while(insert_into_channel(OUTCHANNEL, string) != FULL)
        {
        enable();
        if(*string == '\0')
                return(SUCCESS);
        else
                string++;
        }
enable();
return(NOSUCCESS);
}

extract(a,b,n)   /* looks for ocurrence of first n characters of string b in
                    string a.  Used to parse commands  */
char *a, *b;
int n;
{
register char *aa;
char*temp;
int k;
k = 0;
aa = a;
temp = b;
while( *aa != '\0')
        {
        if( toupper(*aa) == *b )
          {
          aa++;
          b++;
          k++;
          if( k == n )
                return(SUCCESS);
          }
        else
          {
          aa++;
          b = temp;
          k = 0;
          }
        }
return(NOSUCCESS);
}

/*  Start of initialization sequence  */

menu1(command)
```

```
char *command;
{
int menu2();
disp("\r\nINITIALIZE CHARGER");
next_item = menu2;
waitcount = 0;
}

menu2(command)
char *command;
{
int menu1();
int menu3();
if(*command == NOCOMMAND)
        {
        if(waitcount++ == 10)
                next_item = menu1;
        }
else
        {
        disp("\r\nSET PRESENT TIME - DAY: ");
        waitcount = 0;
        next_item = menu3;
        }
}

menu3(command)
char *command;
{
int i;
int menu4();
for(i = 0; i < 7; i++)
        {
        if(extract(command, wkdays[i], 2) == SUCCESS)
          {
          setday(i);
          disp("\r\nHOUR (24 HR CLOCK)? ");
          next_item = menu4;
          return;
          }
        }
if(*command != NOCOMMAND)
        disp("\r\nDAY OF WEEK? ");
}

menu4(command)
char*command;
{
int i;
int menu5();
if(isdigit(*command))
        {
```

```
        if((i = atoi(command)) < 24)
          {
          sethours(i);
          disp("\r\nMINUTES? ");
          next_item = menu5;
          }
        else
          disp("\r\nHOURS (0 TO 23)?");
        }
else
  {
  if(*command != NOCOMMAND)
        disp("\r\nHOURS (0 TO 23)? ");
  }
}

menu5(command)
char *command;
{
int i;
int menu6();
if(isdigit(*command))
        {
        if((i = atoi(command)) < 60)
          {
          setminutes(i);
          disp("\r\nCHARGING START TIME - HOUR? ");
          next_item = menu6;
          }
        else
          disp("\r\nMINUTES (0 TO 59)? ");
        }
else
        {
        if(*command != NOCOMMAND)
           disp("\r\nMINUTES (0 TO 59)? ");
        }
}

menu6(command)
char *command;
{
int i;
int menu7();
if(isdigit(*command))
        {
        if((i = atoi(command)) < 24)
          {
          chrg_time.hours = i;
          disp("\r\nMINUTES? ");
          next_item = menu7;
          }
```

```
        else
          disp("\r\nHOURS (0 TO 23)?");
        }
else
  {
  if(*command != NOCOMMAND)
        disp("\r\nHOURS (0 TO 23)? ");
  }
}


menu7(command)
char *command;
{
int i;
int menu8();
if(isdigit(*command))
        {
        if((i = atoi(command)) < 60)
          {
          chrg_time.minutes = i;
          disp("\r\nEQUALIZATION DAY? ");
          next_item = menu8;
          }
        else
          disp("\r\nMINUTES (0 TO 59)? ");
        }
else
        {
        if(*command != NOCOMMAND)
           disp("\r\nMINUTES (0 TO 59)? ");
        }
}


menu8(command)
char *command;
{
int i;
int menu9();
for(i = 0; i < 7; i++)
        {
        if(extract(command, wkdays[i], 2) == SUCCESS)
          {
          equ_time.days = i;
          disp("\r\nEQUALIZATION START TIME - HR? ");
          next_item = menu9;
          return;
          }
        }
if(*command != NOCOMMAND)
        disp("\r\nDAY OF WEEK? ");
```

```
}


menu9(command)
char *command;
{
int i;
int menu10();
if(isdigit(*command))
        {
        if((i = atoi(command)) < 24)
          {
          equ_time.hours = i;
          disp("\r\nEQU START TIME - MINUTES? ");
          next_item = menu10;
          }
        else
          disp("\r\nHOURS (0 TO 23)?");
        }
else
  {
  if(*command != NOCOMMAND)
        disp("\r\nHOURS (0 TO 23)? ");
  }
}


menu10(command)
char *command;
{
int i;
int menu11();
if(isdigit(*command))
        {
        if((i = atoi(command)) < 60)
          {
          equ_time.minutes = i;
          disp("\r\nBATTERY MENU (PRESS ENTER KEY) ");
          next_item = menu11;
          }
        else
          disp("\r\nMINUTES (0 TO 59)? ");
        }
else
        {
        if(*command != NOCOMMAND)
           disp("\r\nMINUTES (0 TO 59)? ");
        }
}

menu11(command)
char *command;
```

```
{
int menu12();
if(*command != NOCOMMAND)
        {
        disp("\r\nEXIDE EV-106  (Y/N)? ");
        next_item = menu12;
        }
}

menu12(command)
char *command;
{
int menu13();
int menu16();
if(toupper(*command) == 'Y')
        {
        battype = EV106;
        next_item = menu16;
        disp("\r\nNOMINAL BATTERY VOLTAGE? ");
        }
else
        {
        if(*command != NOCOMMAND)
          {
          disp("\r\nEXIDE XPV-23 (Y/N) ");
          next_item = menu13;
          }
        }
}


menu13(command)
char *command;
{
int menu16();
int menu14();
if(toupper(*command) == 'Y')
        {
        battype = XPV23;
        next_item = menu16;
        disp("\r\nNOMINAL BATTERY VOLTAGE? ");
        }
else
        {
        if(*command != NOCOMMAND)
          {
          disp("\r\nGLOBE UNION EV1000 (Y/N) ");
          next_item = menu14;
          }
        }
}
```

```
menu14(command)
char *command;
{
int menu16();
int menu15();
if(toupper(*command) == 'Y')
        {
        battype = EV1000;
        next_item = menu16;
        disp("\r\nNOMINAL BATTERY VOLTAGE? ");
        }
else
        {
        if(*command != NOCOMMAND)
          {
          disp("\r\nGENERIC LEAD-ACID (Y/N) ");
          next_item = menu15;
          }
        }
}

menu15(command)
char *command;
{
int menu16();
if(toupper(*command) == 'Y')
        {
        battype = GENERIC;
        next_item = menu16;
        disp("\r\nNOMINAL BATTERY VOLTAGE? ");
        }
else
        {
        if(*command != NOCOMMAND)
          {
          disp("\r\nEXIDE EV-106 (Y/N) ");
          next_item = menu12;
          }
        }
}


menu16(command)
char *command;
{
int i;
int menu17();
if(isdigit(*command))
        {
        i = atoi(command);
        switch (i) {
```

```
        case 90:
        case 96:
        case 102:
        case 108:
           batvolts = i;
           next_item = menu17;
           disp("\r\nAC SUPPLY RATING (AMPS)? ");
           break;
        default:
           disp("\r\nMUST BE 90, 96, 102, OR 108 ");
           break;
        }
        }
else
        {
        if(*command != NOCOMMAND)
         disp("\r\n ENTER NOMINAL BATTERY VOLTAGE ");
        }
}

menu17(command)
char *command;
{
int i;
int menu18();
if(isdigit(*command))
        {
        if((i=atoi(command)) > 5 && i < 51)
           {
           if(i > 25 )
              ampsin = 25;
           else
              ampsin = i;
           next_item = menu18;
           disp("\r\nCHARGER CONTROL PANEL MENU ");
           initflg = ON;
           }
        else
           disp("\r\nSUPPLY RATING UNREASONABLE - TRY AGAIN");
        }
else
        {
        if(*command != NOCOMMAND)
           disp("\r\nAMPERE RATING OF AC SUPPLY ");
        }
}

/*  Control Panel Menu  Entry*/

menu18(command)
char *command;
{
```

```
int menu19();
int menu21();
int cp1();
switch(toupper(*command)){

        case 'N':
        case 'F':
                disp("\r\nCLOCK/TIMER MENU ");
                next_item = menu19;
                break;
        case 'B':
                disp("\r\nCHARGER INDICATORS MENU ");
                next_item = menu21;
                break;
        case 'X':
                disp("\r\nNO HIGHER LEVEL MENU ");
                break;
        case 'Y':
                if(onoff == ON)
                   disp("\r\nTURN CHARGER OFF ");
                else
                   disp("\r\nTURN CHARGER ON ");
                next_item = cp1;
                break;
        default:
                break;
        }
}

/* Clock/Timer Menu Entry  */

menu19(command)
char *command;
{
int menu20();
int clktim1();
switch(toupper(*command)){

        case 'B':
                disp("\r\nCHARGER CONTROL PANEL MENU ");
                next_item = menu18;
                break;
        case 'N':
        case 'F':
                disp("\r\nCHARGER SETTINGS MENU ");
                next_item = menu20;
                break;
        case 'Y':
                disp("\r\nDISPLAY PRESENT TIME? ");
                next_item = clktim1;
                break;
       case 'X':
```

```
                disp("\r\nNO HIGHER LEVEL MENU ");
                break;
        default:
                break;
        }
}

/*  Charger Settings Menu Entry  */

menu20(command)
char *command;
{
int set1();
int menu21();
switch (toupper(*command)){
        case 'B':
                disp("\r\nCLOCK/TIMER MENU ");
                next_item = menu19;
                break;
        case 'F':
        case 'N':
                disp("\r\nCHARGER INDICATORS MENU ");
                next_item = menu21;
                break;
        case 'Y':
                disp("\r\nCHANGE BATTERY TYPE ? ");
                next_item = set1;
                break;
        case 'X':
                disp("\r\nNO HIGHER LEVEL MENU");
                break;
        default:
                break;
        }
}

/*  Charger Indicators Menu Entry  */

menu21(command)
char *command;
{
int indic1();
switch(toupper(*command)){
        case 'B':
                disp("\r\nCHARGER SETTINGS MENU ");
                next_item = menu20;
                break;
        case 'N':
        case 'F':
                disp("\r\nCHARGER CONTROL PANEL MENU ");
                next_item = menu18;
                break;
```

```
        case 'Y':
                disp("\r\nDISPLAY CHARGER STATE? ");
                next_item = indic1;
                break;
        case 'X':
                disp("\r\nNO HIGHER LEVEL MENU");
                break;
        default:
                break;
        }
}

/*  Control Panel Menu Functions */

cp1(command)
char *command;
{
int cp2();
int cp3();
switch (toupper(*command)){
        case 'B':
                disp("\r\nSPECIAL ");
                next_item = cp3;
                break;
        case 'N':
        case 'F':
                if (nextstate == IDLEOFF)
                        disp("\r\nSTART CHARGING NOW? ");
                else if (nextstate == FAULT)
                        disp("\r\nCHARGING STOPPED - FAULT");
                else
                        disp("\r\nSTOP CHARGING NOW? ");
                next_item = cp2;
                break;
        case 'Y':
                if(onoff == ON)
                        {
                        onoff = OFF;
                        disp("\r\nTURN CHARGER ON ");
                        }
                else
                        {
                        onoff = ON;
                        disp("\r\nTURN CHARGER OFF ");
                        }
                break;
        case 'X':
                disp("\r\nCHARGER CONTROL PANEL MENU ");
                next_item = menu18;
                break;
        default:
                break;
```

```
        }
}

cp2(command)
char *command;
{
int cp3();
switch(toupper(*command)) {
        case 'B':
                if(onoff == ON)
                        disp("\r\nTURN CHARGER OFF ");
                else
                        disp("\r\nTURN CHARGER ON ");
                next_item = cp1;
                break;
        case 'N':
        case 'F':
                disp("\r\nSPECIAL ");
                next_item = cp3;
                break;
        case 'Y':
                if (nextstate == FAULT)
                        {
                        stopnow = ON;
                        startnow = OFF;
                        disp("\r\nFAULT - CHARGER SHUTDOWN");
                        }
                else if (nextstate != IDLEOFF)
                        {
                        stopnow = ON;
                        disp("\r\nSTART CHARGING NOW ");
                        }
                else
                        {
                        startnow = ON;
                        disp("\r\nSTOP CHARGING NOW ");
                        }
                break;
        case 'X':
                disp("\r\nCHARGER CONTROL PANEL MENU ");
                next_item = menu18;
                break;
        default:
                break;
        }
}

cp3(command)
char *command;
{
int sp1();
switch(toupper(*command)){
```

```
        case 'B':
                if(nextstate == FAULT)
                        disp("\r\nFAULT - CHARGER SHUTDOWN");
                else if(nextstate != IDLEOFF)
                         disp("\r\nSTOP CHARGING NOW ");
                     else
                         disp("\r\nSTART CHARGING NOW ");
                next_item = cp2;
                break;
        case 'N':
        case 'F':
                if(onoff == ON)
                  disp("\r\nTURN CHARGER OFF ");
                else
                  disp("\r\nTURN CHARGER ON ");
                next_item = cp1;
                break;
        case 'Y':
                disp("\r\nCOMMAND? ");
                next_item = sp1;
                break;
        case 'X':
                disp("\r\nCHARGER CONTROL PANEL MENU ");
                next_item = menu18;
                break;
        default:
                break;
        }
}

sp1(command)
char *command;
{
switch(toupper(*command)) {
        case 'C':
/*special commands here */
        case 'X':
                disp("\r\nSPECIAL ");
                next_item = cp3;
                break;
        default:
                break;
        }
}


/*  Clock/Timer Menu Functions  */

clktim1(command)
char *command;
{
int clktim6();
```

```
int clktim2();
int ptl();
char string[15];
struct time ptime;
switch(toupper(*command)){
   case 'B':
        disp("\r\nSET EQU TIME? ");
        next_item = clktim6;
        break;
   case 'N':
   case 'F':
        disp("\r\nSET PRESENT TIME? ");
        next_item = clktim2;
        break;
   case 'Y':
        gettime(&ptime);
        timestring(string,ptime.days,ptime.hours,ptime.minutes);
        disp(string);
        next_item = ptl;
        break;
  case 'X':
        disp("\r\nCLOCK/TIMER MENU ");
        next_item = menu19;
        break;
  default:
        break;
  }
}

ptl(command)
char *command;
{
int clktim2();
if (*command != NOCOMMAND)
        {
        disp("\r\nSET PRESENT TIME? ");
        next_item = clktim2;
        }
}

/* timestring assembles a string of characters representing the day
   of week and time of day  */

timestring(s,i,j,k)
char *s;                    /* pointer to string  */
int i, j , k;               /* day of week (0 to 6), hour, minutes */
{
int n;
char *m;
m = wkdays[i];
*s = '\r';
s++;
```

```
for(n = 0; n < 3; n++, s++, m++)
        { *s = *m;}
*s = ' ';
s++;
*s = (j/10) + '0';
s++;
*s = (j%10) + '0';
s++;
*s = ':';
s++;
*s = (k/10) + '0';
s++;
*s = (k%10) + '0';
s++;
*s = '\0';
}

clktim2(command)
char *command;
{
int clktim3();
int ptim2();
switch(toupper(*command)) {
        case 'B':
                disp("\r\nDISPLAY PRESENT TIME? ");
                next_item = clktim1;
                break;
        case 'N':
        case 'F':
                disp("\r\nDISPLAY CHARGE TIME? ");
                next_item = clktim3;
                break;
        case 'Y':
                disp("\r\nDAY? ");
                next_item = ptim2;
                break;
        case 'X':
                disp("\r\nCLOCK/TIMER MENU ");
                next_item = menu19;
                break;
        default:
                break;
        }
}

ptim2(command)
char *command;
{
int ptim3();
int i;
for(i = 0; i < 7; i++)
        {
```

```
        if(extract(command, wkdays[i], 2) == SUCCESS)
          {
          setday(i);
          disp("\r\nHOUR (24 HR CLOCK)? ");
          next_item = ptim3;
          return;
          }
        }
if(*command != NOCOMMAND)
        disp("\r\nDAY OF WEEK? ");
}

ptim3(command)
char *command;
{
int ptim4();
int i;
if(isdigit(*command))
        {
        if((i = atoi(command)) < 24)
          {
          sethours(i);
          disp("\r\nMINUTES? ");
          next_item = ptim4;
          }
        else
          disp("\r\nHOURS (0 TO 23)?");
        }
else
  {
  if(*command != NOCOMMAND)
        disp("\r\nHOURS (0 TO 23)? ");
  }
}

ptim4(command)
char *command;
{
int i;
int clktim3();
if(isdigit(*command))
        {
        if((i = atoi(command)) < 60)
          {
          setminutes(i);
          disp("\r\nDISPLAY CHARGE TIME? ");
          next_item = clktim3;
          }
        else
          disp("\r\nMINUTES (0 TO 59)? ");
        }
else
```

```
        {
        if(*command != NOCOMMAND)
           disp("\r\nMINUTES (0 TO 59)? ");
        }
}

clktim3(command)
char *command;
{
char string[15];
struct time ptime;
int clktim4();
int chrgtim1();
switch(toupper(*command)){
        case 'B':
                disp("\r\nSET PRESENT TIME? ");
                next_item = clktim2;
                break;
        case 'N':
        case 'F':
                disp("\r\nSET CHARGE TIME? ");
                next_item = clktim4;
                break;
        case 'Y':
                gettime(&ptime);
                timestring(string,ptime.days,chrg_time.hours,chrg_time.minutes);
                disp(string);
                next_item = chrgtim1;
                break;
        case 'X':
                disp("\r\nCLOCK/TIMER MENU ");
                next_item = menu19;
                break;
        default:
                break;
        }
}

chrgtim1(command)
char *command;
{
int clktim4();
if(*command != NOCOMMAND)
        {
        disp("\r\nSET CHARGE TIME? ");
        next_item = clktim4;
        }
}

clktim4(command)
char *command;
{
```

```
int clktim5();
int chgtim2();
switch(toupper(*command)){
        case 'B':
                disp("\r\nDISPLAY CHARGE TIME? ");
                next_item = clktim3;
                break;
        case 'N':
        case 'F':
                disp("\r\nDISPLAY EQU. TIME? ");
                next_item = clktim5;
                break;
        case 'Y':
                disp("\r\nHOURS (24 HR. CLOCK)? ");
                next_item = chgtim2;
                break;
        case 'X':
                disp("\r\nCLOCK/TIMER MENU ");
                next_item = menu19;
                break;
        default:
                break;
        }
}

chgtim2(command)
char *command;
{
int chgtim3();
int i;
if(isdigit(*command))
        {
        if((i = atoi(command)) < 24)
          {
          chrg_time.hours = i;
          disp("\r\nMINUTES? ");
          next_item = chgtim3;
          }
        else
          disp("\r\nHOURS (0 TO 23)?");
        }
else
  {
  if(*command != NOCOMMAND)
        disp("\r\nHOURS (0 TO 23)? ");
  }
}

chgtim3(command)
char *command;
{
int i;
```

```
int clktim5();
int clktim6();
if(isdigit(*command))
        {
        if((i = atoi(command)) < 60)
          {
          chrg_time.minutes = i;
          disp("\r\nDISPLAY EQU. TIME? ");
          next_item = clktim5;
          }
        else
          disp("\r\nMINUTES (0 TO 59)? ");
        }
else
        {
        if(*command != NOCOMMAND)
           disp("\r\nMINUTES (0 TO 59)? ");
        }
}

clktim5(command)
char *command;
{
char string[15];
int clktim6();
int eqtim1();
switch(toupper(*command)){
        case 'B':
                disp("\r\nSET CHARGE TIME? ");
                next_item = clktim4;
                break;
        case 'N':
        case 'F':
                disp("\r\nSET EQU. TIME? ");
                next_item = clktim6;
                break;
        case 'Y':
                timestring(string,equ_time.days,equ_time.hours,equ_time.minutes);
                disp(string);
                next_item = eqtim1;
                break;
        case 'X':
                disp("\r\nCLOCK/TIMER MENU ");
                next_item = menu19;
                break;
        default:
                break;
        }
}

eqtim1(command)
char *command;
```

```
{
int clktim6();
if(*command != NOCOMMAND)
        {
        disp("\r\nSET EQU. TIME? ");
        next_item = clktim6;
        }
}

clktim6(command)
char *command;
{
int eqtim2();
switch(toupper(*command)){
        case 'B':
                disp("\r\nDISPLAY EQU. TIME? ");
                next_item = clktim5;
                break;
        case 'N':
        case 'F':
                disp("\r\nDISPLAY PRESENT TIME? ");
                next_item = clktim1;
                break;
        case 'Y':
                disp("\r\nDAY OF WEEK? ");
                next_item = eqtim2;
                break;
        case 'X':
                disp("\r\nCLOCK/TIMER MENU ");
                next_item = menu19;
                break;
        default:
                break;
        }
}

eqtim2(command)
char *command;
{
int i;
int eqtim3();
for(i = 0; i < 7; i++)
        {
        if(extract(command, wkdays[i], 2) == SUCCESS)
          {
          equ_time.days = i;
          disp("\r\nEQU. START TIME - HR? ");
          next_item = eqtim3;
          return;
          }
        }
if(*command != NOCOMMAND)
```

```
        disp("\r\nDAY OF WEEK? ");
}


eqtim3(command)
char *command;
{
int i;
int eqtim4();
if(isdigit(*command))
        {
        if((i = atoi(command)) < 24)
          {
          equ_time.hours = i;
          disp("\r\nEQU START TIME - MINUTES? ");
          next_item = eqtim4;
          }
        else
          disp("\r\nHOURS (0 TO 23)?");
        }
else
  {
  if(*command != NOCOMMAND)
        disp("\r\nHOURS (0 TO 23)? ");
  }
}

eqtim4(command)
char *command;
{
int i;
if(isdigit(*command))
        {
        if((i = atoi(command)) < 60)
          {
          equ_time.minutes = i;
          disp("\r\nDISPLAY PRESENT TIME? ");
          next_item = clktim1;
          }
        else
          disp("\r\nMINUTES (0 TO 59)? ");
        }
else
        {
        if(*command != NOCOMMAND)
           disp("\r\nMINUTES (0 TO 59)? ");
        }
}

/* Charger Settings Menu Functions  */

set1(command)
```

```
char *command;
{
int set3();
int set2();
int bt1();
switch(toupper(*command)){
        case 'B':
                disp("\r\nSET AC SUPPLY RATING? ");
                next_item = set3;
                break;
        case 'N':
        case 'F':
                disp("\r\nSET BATTERY VOLTAGE? ");
                next_item = set2;
                break;
        case 'Y':
                disp("\r\nEXIDE EV-106 (Y/N)? ");
                next_item = bt1;
                break;
        case 'X':
                disp("\r\nCHARGER SETTINGS MENU ");
                next_item = menu20;
                break;
        default:
                break;
        }
}

/*  Battery Type Sub-menu */

bt1(command)
char *command;
{
int set2();
int bt2();
if(toupper(*command) == 'Y')
        {
        battype = EV106;
        next_item = set2;
        disp("\r\nSET BATTERY VOLTAGE? ");
        }
else
        {
        if(*command != NOCOMMAND)
          {
          disp("\r\nEXIDE XPV-23 (Y/N) ");
          next_item = bt2;
          }
        }
}
```

```
bt2(command)
char *command;
{
int set2();
int bt3();
if(toupper(*command) == 'Y')
        {
        battype = XPV23;
        next_item = set2;
        disp("\r\nSET BATTERY VOLTAGE? ");
        }
else
        {
        if(*command != NOCOMMAND)
          {
          disp("\r\nGLOBE UNION EV1000 (Y/N) ");
          next_item = bt3;
          }
        }
}


bt3(command)
char *command;
{
int set2();
int bt4();
if(toupper(*command) == 'Y')
        {
        battype = EV1000;
        next_item = set2;
        disp("\r\nSET BATTERY VOLTAGE? ");
        }
else
        {
        if(*command != NOCOMMAND)
          {
          disp("\r\nGENERIC LEAD-ACID (Y/N) ");
          next_item = bt4;
          }
        }
}

bt4(command)
char *command;
{
int set2();
if(toupper(*command) == 'Y')
        {
        battype = GENERIC;
        next_item = set2;
        disp("\r\nSET BATTERY VOLTAGE? ");
```

```
        }
else
        {
        if(*command != NOCOMMAND)
          {
          disp("\r\nEXIDE EV-106 (Y/N) ");
          next_item = bt1;
          }
        }
}

set2(command)
char *command;
{
int set3();
int bv1();
switch(toupper(*command)){
        case 'B':
                disp("\r\nCHANGE BATTERY TYPE? ");
                next_item = set1;
                break;
        case 'N':
        case 'F':
                disp("\r\nSET AC SUPPLY RATING? ");
                next_item = set3;
                break;
        case 'Y':
                disp("\r\nNOMINAL BATT. VOLTAGE? ");
                next_item = bv1;
                break;
        case 'X':
                disp("\r\nCHARGER SETTINGS MENU ");
                next_item = menu20;
                break;
        default:
                break;
        }
}

bv1(command)
char *command;
{
int i;
int set3();
if(isdigit(*command))
        {
        i = atoi(command);
        switch (i) {
        case 90:
        case 96:
        case 102:
        case 108:
```

```
            batvolts = i;
            next_item = set3;
            disp("\r\nSET AC SUPPLY RATING? ");
            break;
         default:
            disp("\r\nMUST BE 90, 96, 102, OR 108 ");
            break;
         }
         }
else
         {
         if(*command != NOCOMMAND)
          disp("\r\n ENTER NOMINAL BATTERY VOLTAGE ");
         }
}

set3(command)
char *command;
{
int sr1();
switch(toupper(*command)){
         case 'B':
                  disp("\r\nSET BATTERY VOLTAGE? ");
                  next_item = set2;
                  break;
         case 'N':
         case 'F':
                  disp("\r\nCHANGE BATTERY TYPE? ");
                  next_item = set1;
                  break;
         case 'Y':
                  disp("\r\nAC SUPPLY RATING (AMPS)? ");
                  next_item = sr1;
                  break;
         case 'X':
                  disp("\r\nCHARGER SETTINGS MENU ");
                  next_item = menu20;
                  break;
         default:
                  break;
         }
}

sr1(command)
char *command;
{
int i;
if(isdigit(*command))
         {
         if((i=atoi(command)) > 5 && i < 51)
            {
            if(i > 25 )
```

```
                ampsin = 25;
            else
                ampsin = i;
            next_item = set1;
            disp("\r\nCHANGE BATTERY TYPE? ");
            }
        else
            disp("\r\nSUPPLY RATING UNREASONABLE - TRY AGAIN");
        }
else
        {
        if(*command != NOCOMMAND)
            disp("\r\nAMPERE RATING OF AC SUPPLY ");
        }
}

/* Charger Indicators Menu Functions  */

indic1(command)
char *command;
{
int indic4();
int indic2();
int showstate();
switch(toupper(*command)){
        case 'B':
                disp("\r\nBATTERY TEMPERATURE? ");
                next_item = indic4;
                break;
        case 'N':
        case 'F':
                disp("\r\nBATTERY VOLTAGE? ");
                next_item = indic2;
                break;
        case 'Y':
                next_item = showstate;
                waitcount = 0;
                break;
        case 'X':
                disp("\r\nCHARGER INDICATORS MENU ");
                next_item = menu21;
                break;
        default:
                break;
        }
}

showstate(command)
char *command;
{
int indic2();
if(*command != NOCOMMAND)
```

```
        {
        disp("\r\nBATTERY VOLTAGE? ");
        next_item = indic2;
        return;
        }
if(waitcount++ == 0)
        {
        switch(nextstate){
                case IDLEOFF:
                      disp("\r\nIDLE/OFF");
                      break;
                case CONSTCURR:
                        disp("\r\nCONSTANT CURRENT CHARGE");
                        break;
                case CONSTVOLT:
                        disp("\r\nCONSTANT VOLTAGE CHARGE");
                        break;
                case FINISH:
                        disp("\r\nFINISHING CHARGE");
                        break;
                case EQUAL:
                        disp("\r\nEQUALIZING CHARGE");
                        break;
                case BLOCKED:
                        if(blockreason == UNDERTEMP)
                          {
                          disp("\r\nBLOCKED - LOW TEMPERATURE");
                          break;
                          }
                        if(blockreason == NOPOWER)
                          {
                           disp("\r\nBLOCKED - NO POWER");
                           break;
                           }
                        disp("\r\nIDLE/BLOCKED");
                        break;
                case FAULT:
                        if(faultreason == HITEMP)
                                {
                                disp("\r\nFAULT - BATTERY OVERTEMP.");
                                break;
                                }
                        if(faultreason == HIVOLTS)
                                {
                                disp("\r\nFAULT - BATTERY OVERVOLTS");
                                break;
                                }
                        if(faultreason == LOVOLTS)
                                {
                                disp("\r\nFAULT - BATTERY LOWVOLTS");
                                break;
                                }
```

```
                        if(faultreason == OVRTIME)
                                {
                                disp("\r\nFAULT - OVERTIME");
                                break;
                                }
                        else
                        {
                        disp("\r\nFAULT ");
                        break;
                        }
                }
        }
if(waitcount == 10)
        waitcount = 0;
}

indic2(command)
char *command;
{
int indic3();
int showvolts();
switch(toupper(*command)){
        case 'B':
                disp("\r\nCHARGER STATE? ");
                next_item = indic1;
                break;
        case 'N':
        case 'F':
                disp("\r\nCHARGE CURRENT? ");
                next_item = indic3;
                break;
        case 'Y':
                waitcount = 0;
                next_item = showvolts;
                break;
        case 'X':
                disp("\r\nCHARGER INDICATORS MENU ");
                next_item = menu21;
                break;
        default:
                break;
        }
}

showvolts(command)
char *command;
{
int indic3();
char vstring[5];
if(*command != NOCOMMAND)
        {
        disp("\r\nCHARGE CURRENT? ");
```

```
        next_item = indic3;
        return;
        }
if(waitcount++ == 0)
        {
        itoa(dvolts,vstring);
        disp(vstring);
        }
if(waitcount == 10)
        waitcount = 0;
}

indic3(command)
char *command;
{
int indic4();
int showcurrent();
switch(toupper(*command)){
        case 'B':
                disp("\r\nBATTERY VOLTAGE? ");
                next_item = indic2;
                break;
        case 'N':
        case 'F':
                disp("\r\nBATTERY TEMPERATURE? ");
                next_item = indic4;
                break;
        case 'Y':
                waitcount = 0;
                next_item = showcurrent;
                break;
        case 'X':
                disp("\r\nCHARGER INDICATORS MENU ");
                next_item = menu21;
                break;
        default:
                break;
        }
}

showcurrent(command)
char *command;
{
int indic4();
char cstring[5];
if(*command != NOCOMMAND)
        {
        disp("\r\nBATTERY TEMPERATURE? ");
        next_item = indic4;
        return;
        }
if(waitcount++ == 0)
```

```
        {
        itoa(dcurrent,cstring);
        disp(cstring);
        }
if(waitcount == 10)
        waitcount = 0;
}

indic4(command)
char *command;
{
int showtemp();
switch(toupper(*command)){
        case 'B':
                disp("\r\nCHARGE CURRENT? ");
                next_item = indic3;
                break;
        case 'N':
        case 'F':
                disp("\r\nCHARGER STATE? ");
                next_item = indic1;
                break;
        case 'Y':
                waitcount = 0;
                next_item = showtemp;
                break;
        case 'X':
                disp("\r\nCHARGER INDICATORS MENU ");
                next_item = menu21;
                break;
        default:
                break;
        }
}

showtemp(command)
char *command;
{
char tstring[5];
if(*command != NOCOMMAND)
        {
        disp("\r\nCHARGER STATE? ");
        next_item = indic1;
        return;
        }
if(waitcount++ == 0)
        {
        itoa(dtemp,tstring);
        disp(tstring);
        }
if(waitcount == 10)
        waitcount = 0;
```

CA1
PTC 8D1
86
A 25F
c.2
(MM)

```
}

itoa(n,s)
char s[];
int n;
{
int i,j;
s[0]='\r';
i = 1;
do{
s[i++] = (n%10) + '0';
  }
while((n/=10) > 0);
s[i] = '\0';
for (j = 1, i--; j < i; j++, i--)
        {
        n = s[j];
        s[j] = s[i];
        s[i] = n;
        }
}
```

Library and Information Centre / Centre de documentation (GSLA)
TRANSPORT CANADA
OCT 8 1986
Transport Canada / Transports Canada